

iterations

vicky isley & paul smith
www.boredomresearch.net
aoc@boredomresearch.net

introduction

This session will cover the following:

AM

- Students present their week four assignments.
- Paper exercise to demonstrate if and else structure and then show an example of these conditions in Processing.
- Introduction using relational operators, for structures and loops in Processing.
- Practical workshop where students swap their Processing tools and build a composition combining their functions.
- Exercise to illustrate the random function.

PM

- Presentation on “The Process of Generative Compositions.”
- Exercise to illustrate how to save static images & frames from Processing compositions.
- Demonstrate good methods of organising your code.
- Students build different compositions and save a series of images.
- Present next week’s assignment.

Present results from week four assignment.

Discuss the different variables and functions students have produced and any problems.

Exercise: what we can do with if and else...

Students execute a paper based drawing exercise that is based on a conditional test.

Participants will write a set of conditions where the response is either true or false.
i.e. You often cry when you watch a sad film? Or you bite your finger nails?

Once they have a set of conditions they will pass these onto a classmate.

The participant will start with drawing a node on the bottom of the paper. The first branch will start from this node.

Each condition will be read and dependant on whether the conditional response is true or false participants will flip their templates accordingly. Drawing branches from the last nodes drawn.

Control the flow of programs

The previous exercise will then be illustrated in Processing through illustrating writing conditional tests.

The key element of a selection structure is the *conditional test*, a shortcut for writing an **if()** and **else** structure. A conditional test is an expression that results in a *boolean* value -- it is either **true** or **false**. If the **condition** evaluates to **true**, **expression1** is evaluated and returned. If the **condition** evaluates to **false**, **expression2** is evaluated and returned.

The following conditional: `condition : expression1 ? expression2`

is equivalent to this structure:

```
if(condition) {  
  expression1  
} else {  
  expression2  
}
```

We can use the following operators in a conditional test:

Relational Operators

> (greater than)
< (less than)
>= (greater than or equal to)
<= (less than or equal to)
== (equality)
!= (inequality)

Logical operators

|| (logical OR)
&& (logical AND)
! (logical NOT)

Relational Operator examples:**>= (greater than or equal to)**

```
int a = 23;
int b = 23;
if(a >= b) {
    println("variable a is greater or equal to variable b ")
}
```

!= (inequality)

```
int a = 22;
int b = 23;
if(a != b) {
    println("variable a is not equal to variable b");
}
```

== (equality)

```
int a = 23;
int b = 23;
if(a == b) {
    println("variables a and b are equal");
}
```

Logical Operator Examples:

|| (logical OR)

Compares two expressions and returns true if one or both evaluate to true. Returns false only if both expressions are false. The following list shows all possible combinations:

```
true || false // Evaluates true because the first is true
false || true // Evaluates true because the second is true
true || true // Evaluates true because both are true
false || false // Evaluates false because both are false
```

! (logical NOT)

Inverts the Boolean value of an expression. Returns **true** if the expression is **false** and returns **false** if the expression is **true**. If the expression **(a>b)** evaluates to true, then **!(a>b)** evaluates to false.

&& (logical AND)

Compares two expressions and returns true only if both evaluate to true. Returns false if one or both evaluate to false. The following list shows all possible combinations:

```
true && false // Evaluates false because the second is false
false && true // Evaluates false because the first is false
true && true // Evaluates true because both are true
false && false // Evaluates false because both are false
```

for() structure

Controls a sequence of repetitions. A **for()** structure has three parts: **init**, **test**, and **update**. Each part must be separated by a semi-colon ";".

For example:

```
for(init; test; update) {  
    statements  
}
```

```
for(int i=0; i<40; i=i+1) {  
    line(30, i, 80, i);  
}
```

The loop continues until the test evaluates to **false**. When a **for()** structure is executed, the following sequence of events occurs:

1. The init statement is executed
2. The test is evaluated to be true or false
3. If the test is true, jump to step 4. If the test is False, jump to step 6
4. Execute the statements within the block
5. Execute the update statement and jump to step 2
6. Exit the loop.

Presentation on “Different output of Generative Compositions.”

Two examples were shown in class of outputting generative compositions created in Processing as static imagery and a series of frames and combining these with photography and film.

Metropop Denim

(<http://www.processing.org/exhibition/works/metropop>)

Fashion Photography and Generative Artwork by Clayton Cubitt and Tom Carden



Tom Carden used the alpha version of Processing to produce a series of applets loosely based around the theme of attractors and particles. Working from a single code-base for particles operating under a gravity-like force, the presentation varied from smoke and spark effects to almost photographic abstract pixel exposures. One branch of the work was the result of an attempt to make a more ethereal and sketch-like visualisation of the forces at work.

Star Nursery

(<http://www.motiontheory.com>)

Ryan Alexander created stars that reacted to video input and utilised this sketch within REMS 'animal' video.



Saving static images & frames in Processing

save() function

Saves an image from the display window. Images are saved in TIFF or TARGA format depending on the extension of the **filename** parameter. If no extension is included in the filename, the image will save in TIFF format and **.tif** will be added to the name. Images are saved to to the "current" folder. For example, when running your sketch from the Processing application, the image will be saved to the location of the Processing application. To save inside the current sketch's folder, use **saveFrame()**. It is not possible to use **save()** while running the program in a web browser.

```
save("diagonal.tif");  
// Saves a TIFF file named "diagonal.tif"
```

saveFrame() function

```
saveFrame("line-####.tif");  
// Saves each frame as line-0000.tif, line-0001.tif, etc.
```

Saves a numbered sequence of images, one image each time the function is run. To save an image identical to the display window, run the function at the end of **draw()** or within mouse and key events such as **mousePressed()**. If **saveFrame()** is called without parameters, it will save the files as screen-0000.tif, screen-0001.tif, etc. It is possible to specify the name of the sequence with the **filename** parameter and make the choice of saving TIFF or TARGA files with the **ext** parameter. These image sequences can be loaded into programs such as Apple's QuickTime software and made into movies. These files are saved to the sketch's folder, which may be opened by selecting "Show sketch folder" from the "Sketch" menu. It is not possible to use **save()** while running the program in a web browser.

Keeping your code organised and readable.

Remember to consider the following:

- What's a good way to name my variables to make my code readable?
- What data type is appropriate for each variable?
- Where do I need a global variable vs. a local variable?
- Where do I initialize my variables?
- What's a good way to organize the various pieces of code? (consider separating your drawing code from your code that does the *calculation* tasks.)

This weeks assignment:

Following on from today's workshop please build your own dynamic generative composition in Processing - using a multitude of different functions written by yourself and classmates. The objective of this exercise is to think about how you can creatively interpret the available tools. Also, to think about how your composition unfolds over time. You may like to consider the possibilities of if else structures and for loops to extend your existing creative potentials along with the possibilities of randomness, interaction and accumulation.

Name the sketch file with your initials and w5a e.g. my file would be called psw5a and the processing code file would be called psw5a.pde.

Save images of your generative composition through different stages. Select three images that document the work well and convert to jpeg format.

Please complete and email your final compositions(s) as .pde files and your three images to us at aoc@boredomreseach.net by Friday 10th March 2006

Outcomes

By the end of the session students will:

- [1] understand the significance of “if argument” and “for loop” programming structures.
- [2] Have experienced a clear illustration of how an if argument can be used to generate graphic output.
- [3] be able to write and implement if.. else structure
- [4] Understand how to implement relational operators.
- [5] Understand how to implement logical operators.
- [6] be able to write and implement a “for loop”.
- [7] Have an appreciation for the use of random to create multiple composition permutations.
- [8] Understand the importance of variable placement for “global” and “local” use.
- [9] be able to output static and dynamic compositions as image files or frames for use as animated sequences.
- [10] be inspired to create dynamic a static computational works from simple software tools.