A New Form of Control – The Loop

We have used one form of control – the if/else. This is a form of control called selection or branching.

But first, a word from our sponsor – arithmetic. . .

We have used arithmetic for everything since homework #2.

We have used + - * and /.

We strongly advised you to avoid int values and use float for some vague reasons about possible "problems." It is now time to look at those possible "problems" and widen your arithmetic horizons a bit.

Here is where the problem comes into play:

```
   int + int = int
  int – int = int
  int * int = int
  int / int = int
```

For the first three, there are no problems with using int. It is division that can haunt you. Let's suppose you are using an int variable along with Processing's width variable which is also an int. Let's further assume that you want to compute a percentage of the width by dividing mouseX (an int value)by the width. This is perfectly logical so you code the following:

```
  float pct = mouseX / width;
```

This is syntactically perfect!

Let's further assume that the width is 400 and mouseX is 200. Your expected answer is .5 or 50%.

It is not! The result is 0! The reason is that an int divided by an int must evaluate to an int. And .5 is not an int!

This next statement is logically correct, syntactically correct but it does not give the expected result:

```
    1 / 4  * width;
```

Assuming that width still has a value of 400, the expected result is 100 but it is not . The result is 0!

The evaluation of division and multiplication have the same precedence so the evaluation proceeds from left to right – the first evaluation is:

```
 1   /   4
```

which evaluates to 0 for the same reason: int divided by int evaluates to an int.

This is why we advised you to use float values. It is also the case that both random( ) and dist( ) return float values.

You can put an int value into a float variable but you cannot directly put a float value into an int variable.

If you have to put a float value into an int variable, you must use function int( ). This function truncates the decimal value of the float. These expressions:

```
  float f = 9.9;
   int i = int( f );
```

gives the variable, i a value of 9. The .9 is lost.

We still strongly advise you to use floats to locate positions in your drawing. The best use of int values (to date) is for counting events that occur while your program is running.

The following variables are int variables:
`width,  height,  frameCount,`
`mouseX, mouseY, pmouseX, pmouseY`

The following function returns an int value:
`millis( );`

And – what about the remainder of division?

So far this set of notes has blathered on about 3 / 4 evaluating to 0 which is the quotient. But what about the remainder?

If we return deep into the past of your arithmetic education – before the dawn of fractions or decimals, we may find that you were taught to do division in a manner similar to this:

```
                  0  r  3
                 ________
               4) 3
                  0
                 __
                  3
```

Maybe not, but this is how Jim was taught. The 0 on the top line is the quotient and the 3 on the top line is the remainder. It is what remains when the division is finished.

The / operator gives us the quotient -- ok – we have said that.
There is another arithmetic operator that we have ignored:

%

This is a percent sign but it has absolutely nothing to do with percentages when it is used to in an arithmetic expression.

Used like this:

```
   4    %    3
```

the expression evaluates to the remainder. You may be thinking, "who cares?" but it turns out that this operator is very useful in several situations. Before we look at them, we need to understand what values a remainder can have. Did you ever think about this? Probably not and there was no reason to do so. But as a programmer, it is worth a bit of text to do so.

Let's stay with the divisor, 4. What is the complete set of possible remainders for any int values divided by 4? Please do not turn your mind off at this point. Stay with this set of notes for a bit longer. The question may seem unreasonable but … maybe not. Here are a few examples:

0	%	4	=	0	8	%	4	=	0
1	%	4	=	1	9	%	4	=	1
2	%	4	=	2	10	%	4	=	2
3	%	4	=	3	11	%	4	=	3
4	%	4	=	0	12	%	4	=	0
5	%	4	=	1	13	%	4	=	1
6	%	4	=	2	14	%	4	=	2
7	%	4	=	3	15	%	4	=	3

There is an interesting and very useful pattern to the value of the remainders. The remainder is a value between 0 and 3. A more general statement for all division expressions with int values that use the % operator is that the result is always a value between 0 and one less than the divisor.

Here are a few ways we can use this:

Tossing a Coin:

Let's suppose we need to toss a coin to determine what to do. If we can find an changing int value we can do this with the % operator. The frameCount is an int so we can do this:

```
    if ( frameCount % 2  == 0 )
    {
       // call it even and so something here
    }
    else
    {
       // call it odd and do a different something here
    }
```

Keeping a moving figure on screen:

Suppose we are using the frameCount variable to determine the horizontal location of a rectangle in a window that is 400 pixels wide:

```
  rect( frameCount, width/2,  30, 30 );
```

When `frameCount` reaches a value of 400, the rectangle disappears.
What about this code?

```
  rect( frameCount % width, width/2,  30, 30 );
```

If you are not sure what happens, put this piece of code in a draw( ) function and see what happens.

```
void draw( )
{
    background( 0 );
    rect( frameCount % width, width/2,  30, 30 );
}
```

There are other very useful situations where we can limit the range of a variable's values using the % operator. Frequently we can use this in place of an if/else. Keep this in mind as we move into the second third of the course.

Now back to our regularly schedule program.

A New Form of Control – The Loop

A second group of control structures is the iterative group or the loops. The term loop is used because the syntax forces Processing to loop over a set of function calls until some condition is met. There are three forms of iteration available to Processing:

- the while loop
- the for loop
- the do loop

Processing ignores the do loop and so will we .

The first loop we will work with is the while loop which has this form.

The API tells us the following:

Name while

Examples

```
int i=0;
while(i<80) {
  line(30, i, 80, i);
  i = i + 5;
}
```

Description

Controls a sequence of repetitions. The **while** structure executes a series of statements continuously while the **expression** is **true**. The expression must be updated during the repetitions or the program will never "break out" of **while**.

This function can be dangerous because the code inside the while() loop will not finish until the expression inside while() becomes true. It will lock out all other code from running (mouse events will not be updated, etc.) So be careful because this can lock up your code (and sometimes even the Processing environment itself) if used incorrectly.

Syntax

```
 while (expression)    // NOTE NO ; GOES HERE
{   statements }
```

Parameters

expression	a valid expression
statements	one or more statements

The statndard textbook explanation looks like this:

```
while ( boolean expression )
{
   // the loop body is the code inside the two braces
}
```

A better explanation might look like this

```
while ( this evaluates to true )
{
   Do all of the stuff in here.
   When this is done,
        go back to the boolean expression and re-evaluate it
}
```

Here is the textbook naming of the parts of the while

```
while ( loop guard or loop test – a boolean expression )
{
    // The loop BODY
    // It is very important that some code in these braces results
    // in the loop guard or loop test eventually evaluating to false
    // or the loop runs infinitely!

}
```

As long as the loop guard or test is true, the body of the loop is executed. Once the loop guard or test evaluates to false, the execution of the loop body stops.

Look at the corresponding class code for this set to see some examples of the use of the while loop.

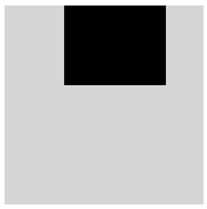
The for Loop:

The Processing API explains the for loop as shown below:

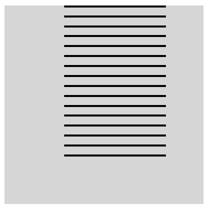
Name

for

Examples



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



```
for (int i = 0; i < 80; i = i+5) {
  line(30, i, 80, i);
}
```

Description

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

Syntax

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

Parameters

init	statement executed once when beginning loop
test	if the test evaluates to true, the statements execute
update	executes at the end of each iteration
statements	collection of statements executed each time through the loop

```
for (  #1 the init ; #2 the test or guard;  #4 the increment or update )
{
   #3 the loop body
   Do all of the stuff in here.
   When this is done, go  to the increment or update.
}
```

The for loop executes the component parts in this order:

#1 ➔ #2 if **`true`** ➔ #3 ➔ #4 ➔ #2 if **`true`** ➔ #3 ➔ #4 ➔ #2 if **`true`** ➔ #3 ➔ #4 ➔ #2 and so on until #2 is **`false`**.

The parts of the **`for`** loop are found in the while loop. The main difference is that the for loop provides specific syntax places to do the init, the test and the update while the while loop has a syntax place for only the test .

The general guideline for choosing which loop to use is:

- If you know the number of iterations needed – use the for loop.
- If you do not know the number of iterations needed – use the while loop.

In reality, either works for any situation.

You need to be able to use both.