Control 1: Repetition

► Syntax / Function Introduced

`for`

► Iteration

Computer are excellent at executing repetitive tasks accurately and quickly. With the conditional structures we learn in last units, we can using Processing to draw repetitive shapes easily.

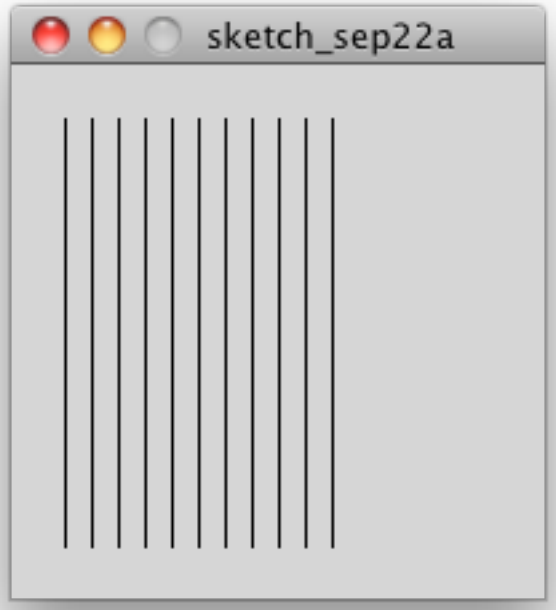
Compare the following codes:

example 1:

```
size(200,200);
line(20, 20, 20, 180);
line(30, 20, 30, 180);
line(40, 20, 40, 180);
line(50, 20, 50, 180);
line(60, 20, 60, 180);
line(70, 20, 70, 180);
line(80, 20, 80, 180);
line(90, 20, 90, 180);
line(100, 20, 100, 180);
line(110, 20, 110, 180);
line(120, 20, 120, 180);
```

example2:

```
size(200,200);
for (int i=20; i<=120; i+=10) {
  line(i, 20, i, 180);
}
```



Both program do the same thing. The first program uses 14 lines of code, but with the for structure, only 4 lines are used in the second program.

Here is the general case of the for structure:

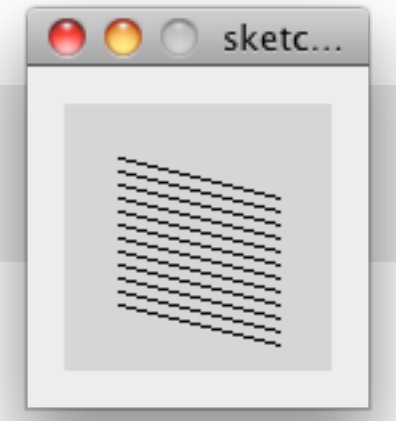
```
for (init; test; update) {
  statements
}
```

The parentheses enclose three items: **`init`**, **`test`** and **`update`**. The statements inside the blocks run continuously while the test evaluates to true. The init part usually assigns an initial value to the variables used in the test. The update is used to modify the value of that variable **AFTER** each iteration through statements in the block.

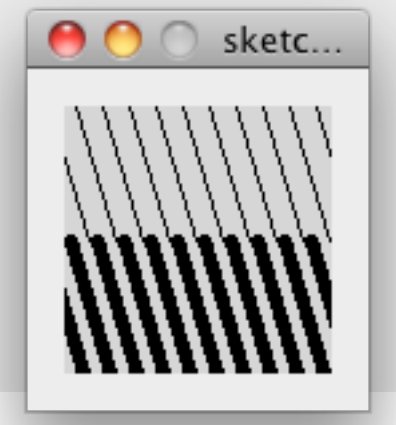
A for structure runs in the following order:

1. The **`init`** statement is run
2. The **`test`** is evaluated to true of false
3. If the **`test`** is true, goto step 4. Otherwise goto step 6
4. Run the statements within the block
5. Run the **`update`** statement and goto step 2
6. Exit the structure and continue running the program

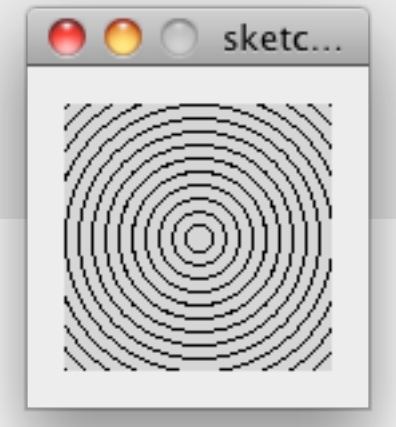
Here are some examples about for structure:



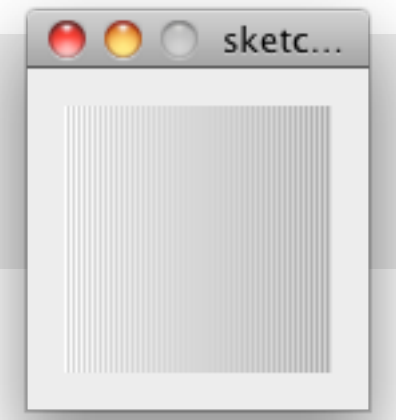

```
for (int i=20; i<80; i+=5) {
  line(20, i, 80, i+15);
}
```




```
for (int x = -16; x<100; x+=10) {
  line (x, 0, x+15, 50);
}
strokeWeight(4);
for (int x = -18; x<100; x+=10) {
  line (x, 50, x+15, 100);
}
```




```
noFill();
for (int d = 150; d>0; d-= 10) {
  ellipse(50,50,d,d);
}
```




```
for (int i=0; i<100; i+=2) {
  stroke(255 - i);
  line(i,0,i,100);
}
```

► Formatting Blocks

There are different styles to format code blocks, the point is to make the code **clear** and **easy to read**:

```
int x = 50;
if (x > 100) {
  line (20, 20, 80, 80);
} else {
  line (80, 20, 20, 80);
}
```

OR

```
int x = 50;
if (x > 100)
{
  line (20, 20, 80, 80);
}
else
{
  line (80, 20, 20, 80);
}
```

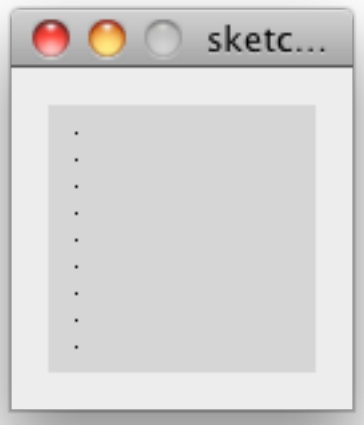
But avoid formatting the code like this:

```
int x = 50;
```

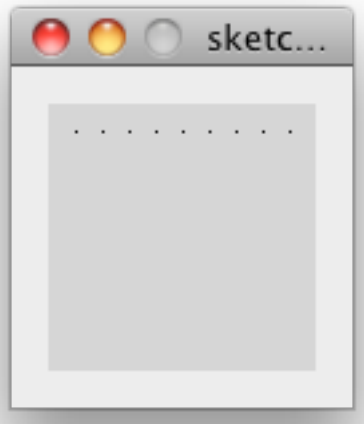
```
if (x > 100){
line (20, 20, 80, 80);
} else {
line (80, 20, 20, 80);
}
```

▶ Formatting Blocks

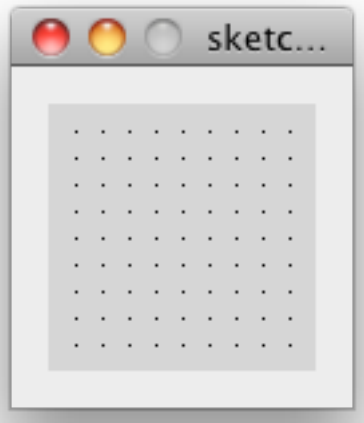
We can put a **`for`** structure into another **`for`** structure, which makes a **nested Iteration** letting you to draw 2D structure shapes.




```
for (int y=10; y<100; y+=10) {
  point(10, y);
}
```

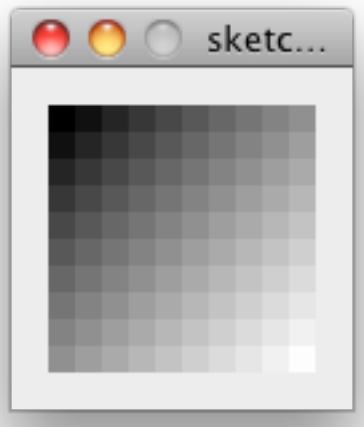



```
for (int x=10; x<100; x+=10) {
  point(x, 10);
}
```

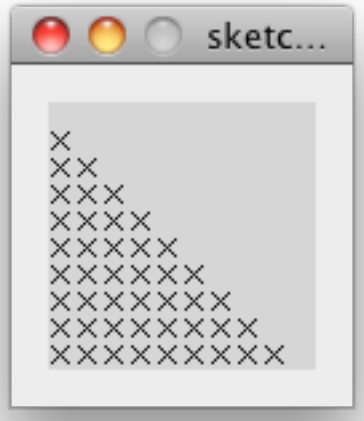



```
for (int y=10; y<100; y+=10) {
  for (int x=10; x<100; x+=10) {
    point(x, y);
  }
}
```

This technique is useful for creating patterns and effects, the for structure can control color, position, size and other visual properties.




```
noStroke();
for (int y=0; y<100; y+=10) {
  for (int x=0; x<100; x+=10) {
    fill((x+y) * 1.4); // change the fill
color
    rect(x,y, 10,10);
  }
}
```




```
for (int y=1; y<100; y+=10) {
  for (int x=1; x<y; x+=10) {
    line(x, y, x+6, y+6);
    line(x+6, y, x, y+6);
  }
}
```

And more possible patterns:

► Task

a) Make your 1-D pattern using one **`for`** structure

b) Try to create a 2-D pattern, using multiple for structures or a nested for structure.