IAT 265

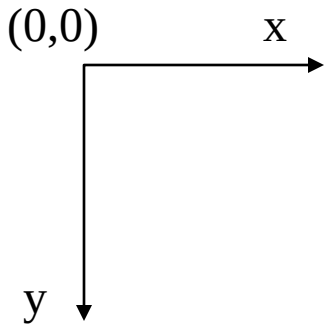
Conditionals, Translation, Rotation

Week 2's Topics

- Conditionals
  - *If*-statement, *if-else*, *nested if-else*
  - *Compound if*-statement
- Translation and Rotation
- Case study: boundary detection, translate & rotate a ladybug

What have we gone over?

- Window size and coordinate system
- Commenting code
- Variables
- Drawing primitives
  - point, line, triangle
  - rect, ellipse
  - arc, curve
- print() and println()
- setup() and draw() – callback methods



Example of animation with setup()/draw()

```
int x;
int y;

void setup() {
   size(400, 400);
   background(0);
   x = 0;
   y = height/2;
}

void draw() {
   background(0);
   ellipse(x, y, 20, 20);
   x = x + 1;
}
```

if-statement

`if` statements introduce conditional execution with syntax:

```
if ( <condition> )
{
  // do this code
}
```

<condition> :
- <boolean vaiable> : holds →
- <boolean expression> : evaluates to →

a boolean value: **`true`** or **`false`**

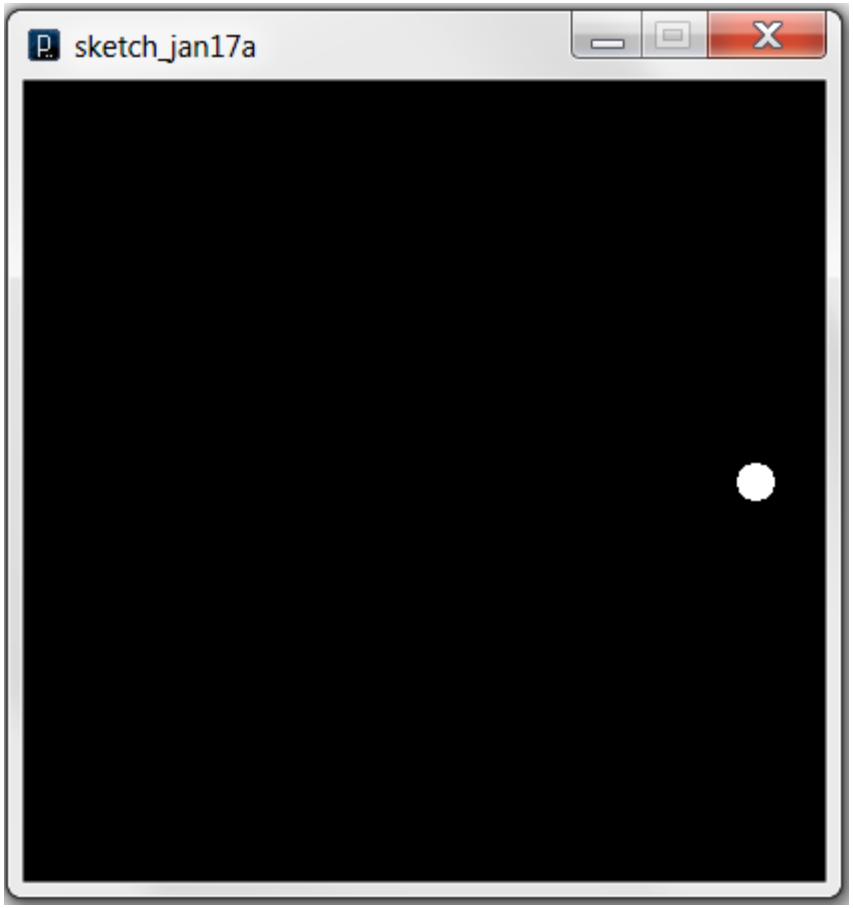
Example:
if with *boolean variable*

```
boolean drawRect = true;

if (drawRect)
{
    fill( 0, 200, 0 ); // fill with green
    rect( 30, 30, 40, 40 );
}

line( 0, 0, 100, 100 );
line( 100, 0, 0, 100 );
```

Try changing the values of drawRect

Example: *if* with *boolean expression*

```
int x;
int y;

void setup() {
   size(400, 400);
   background(0);
   x = 0;
   y = height/2;
}

void draw() {
   background(0);
   ellipse(x, y, 20, 20);
   x = x + 1;
   if(x > width){
        x = 0;
   }
}
```



Relational Operators for Numerical Comparison

TABLE 3.3 NUMERICAL COMPARISON OPERATORS

Operator	Syntax	Description
<	x < y	Result is true if x is *less than* y; otherwise, it is false.
<=	x <= y	Result is true if x is *less than or equal to* y; otherwise, it is false.
>	x > y	Result is true if x is *greater than* y; otherwise, it is false.
>=	x >= y	Result is true if x is *greater than or equal to* y; otherwise, it is false.
==	x == y	**true if x *equals* y; otherwise false**
!=	x != y	**true if x *doesn't equal* y; otherwise false**

The *if-else* Statement

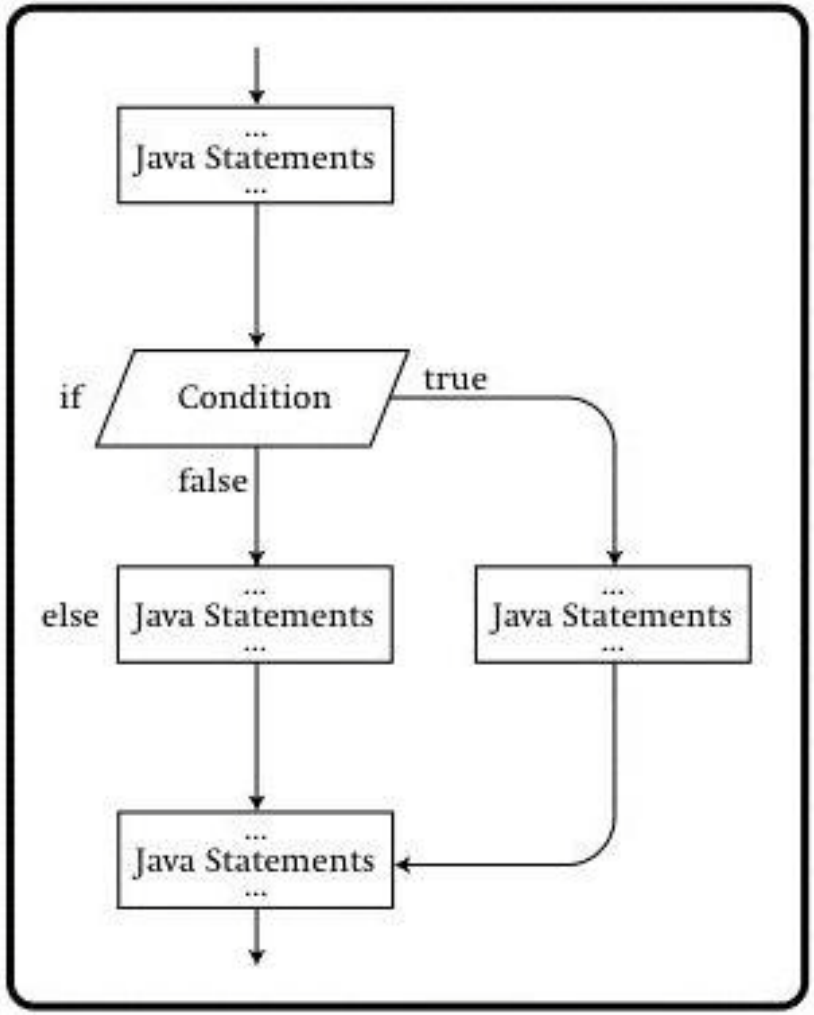
- The *if*-statement alone only allows you to conditionally execute one block of statements
  - The program then executes any statements that follow the *if-block* unconditionally
- What if you want to do something differently if the condition fails?

Flowchart of the *if-else* Structure

- The *if-else* structure allows for branching code into two blocks: executing one or the other

- The syntax:

```
if (condition) {
    statements_for_true;
} else {
    statements_for_false;
}
```



Example *if–else*

```
boolean drawRect = true;

if (drawRect) {
    fill( 0, 200, 0 ); // fill with green
    rect( 30, 30, 40, 40 );

} else {
   ellipseMode(CORNER); //Draw the ellipse from the upper-
                        //left corner of its bounding box
   fill( 0, 200, 220 ); // fill with cyan
   ellipse( 30, 30, 40, 40);
}

line(   0, 0, 100, 100 );
line( 100, 0,   0, 100 );
```

Nested if-else

- Sometimes you need to have more than two branches in the flow of your program based on a set of related conditions
- That is where you can use the *nested if-else* statements. The syntax for it is as follows:

```
if (condition1) {
    statements_for_true_condition1;
}
else if (condition2) {
    statements_for_true_condition2;
}
else { //if all conditions evaluate to false
    statements_for_false_conditions;
}
```

Example *Nested if-else*

```
boolean drawRect = true;
boolean drawEllipse = true;

if (drawRect) {
    fill( 0, 200, 0 ); // fill with green
    rect( 30, 30, 40, 40 );
}
else if( drawEllipse ){
    ellipseMode(CORNER); // Draw the ellipse from the upper-
                         // left corner of its bounding box
    fill( 0, 200, 220 ); // fill with cyan
    ellipse( 30, 30, 40, 40);
}
else { //if all above false
    triangle( 30, 30,  30, 80,  80, 30 );
}
line(   0, 0, 100, 100 );
line( 100, 0,   0, 100 );
```

Compound Conditions

- You can use && or || operator to form compound conditions
- For &&: both sides of it must be true (i.e. *true && true*) for the result to be true
- Example:

```
boolean drawRect = true;
boolean drawInGreen = false;

if (drawRect && drawInGreen) {
    fill( 0, 200, 0 ); // fill with green
    rect( 30, 30, 40, 40 );
}
```

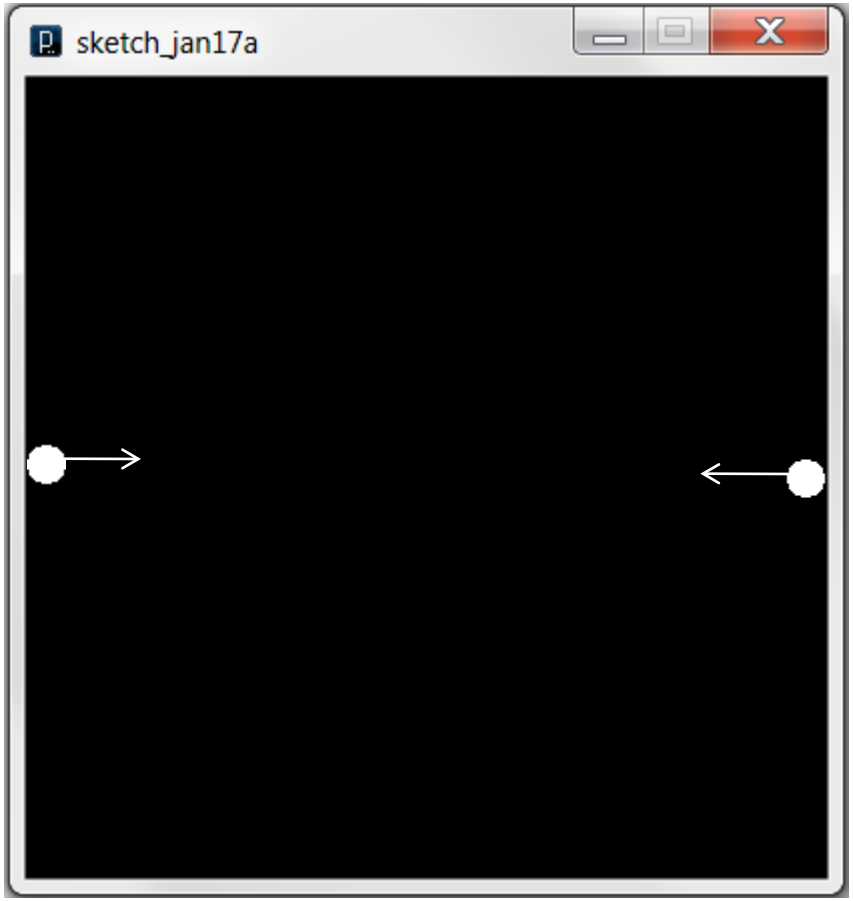
Compound Conditions (2)

- For ||: at least one side of it must be true (i.e. *true* || *true*, *true* || *false*, or *false* || *true*) for the result to be true
- Example:

```
boolean drawRect = true;
boolean drawInGreen = false;

if (drawRect || drawInGreen) {
    fill( 0, 200, 0 ); // fill with green
    rect( 30, 30, 40, 40 );
}
```

Example: Compound *if* with ||

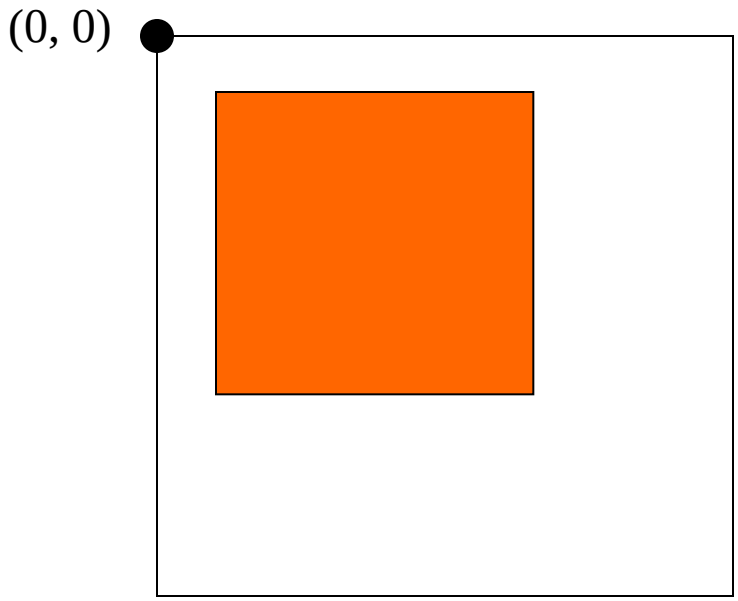
```
int x;
int y;
int changX = 1;
void setup() {
   size(400, 400);
   background(0);
   x = 0;
   y = height/2;
}

void draw() {
   background(0);
   ellipse(x, y, 20, 20);
   x = x + changeX;
   if(x > width || x < 0){
      changX = changeX*-1;
   }
}
```



Translation

- Translation gives us another way of drawing in a new location. It in essence, moves the point (0, 0) in our window.

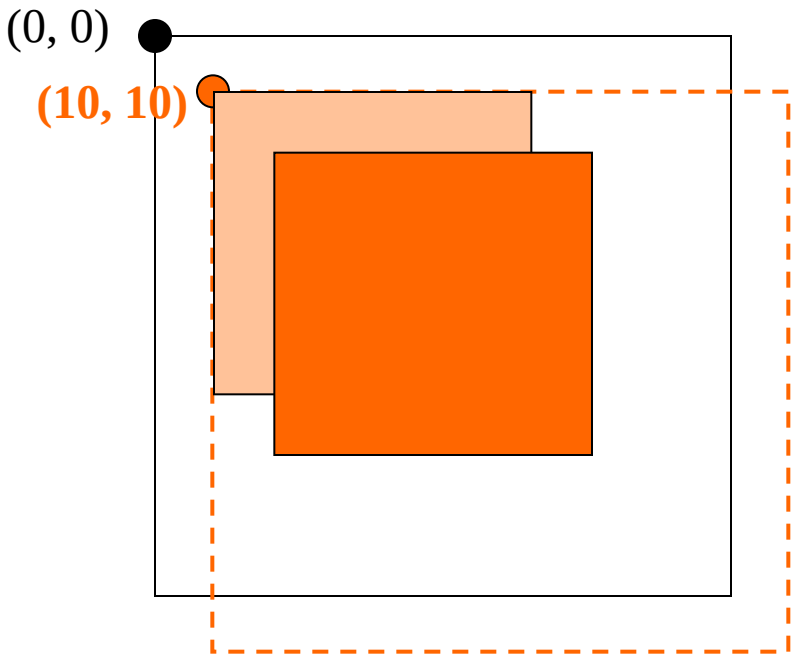
```
fill(255, 128, 0);
rect(10, 10, 50, 50);
```



Translation

- After the call to `translate()`, any drawing functions called will treat our new origin point as if it were (0, 0).

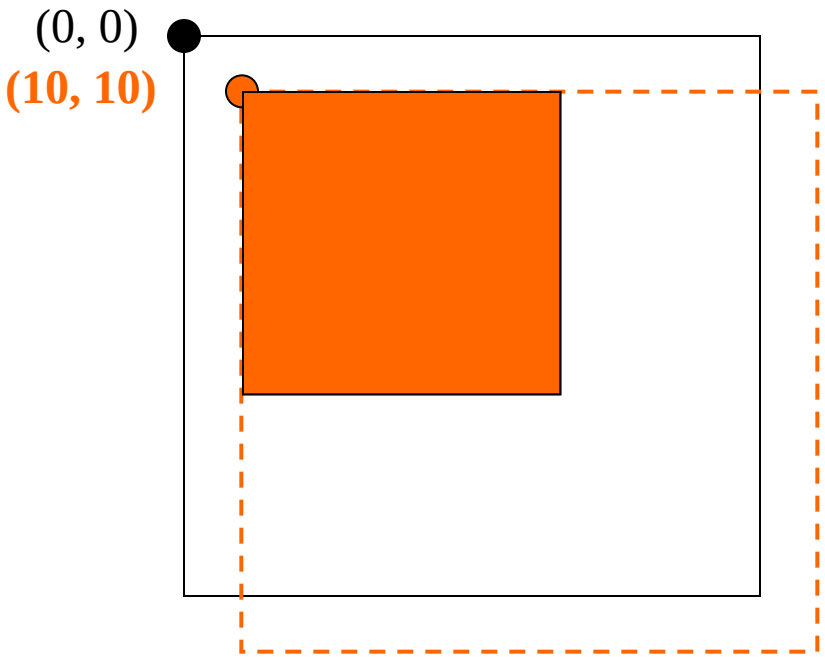
```
//To draw the rect using the same
//coordinates after translation
translate( 10, 10 );
rect(10, 10, 50, 50);
```



Translation (2)

- What if we want to, after the call to `translate(10, 10)`, draw at the same location as before the translation?

```
//To draw the rect at the same
//location as before translation
translate( 10, 10 );
rect( 0, 0, 50, 50);
```



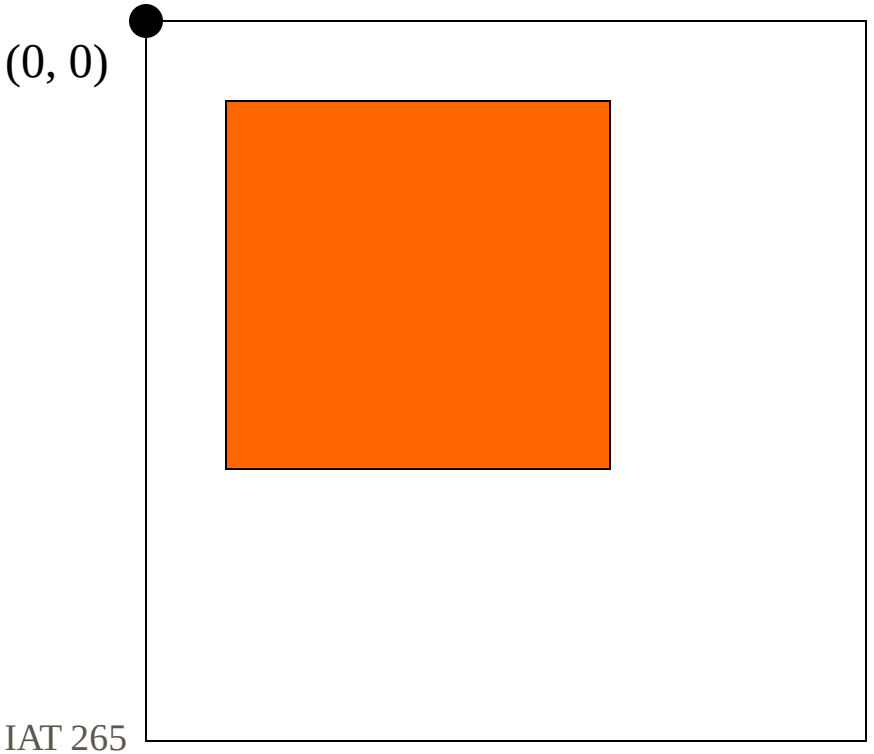
Rotation

- Much like Translation, Rotation moves our drawing space, so that we can draw at different angles.
- Most of the time, you'll want to use Rotation in conjunction with Translation, because `rotate()` rotates the drawing window around the point (0, 0) by default
  - This may not be what you want!!

Rotation

- Let's look at an example without translation:

```
rect(10, 10, 50, 50);
```

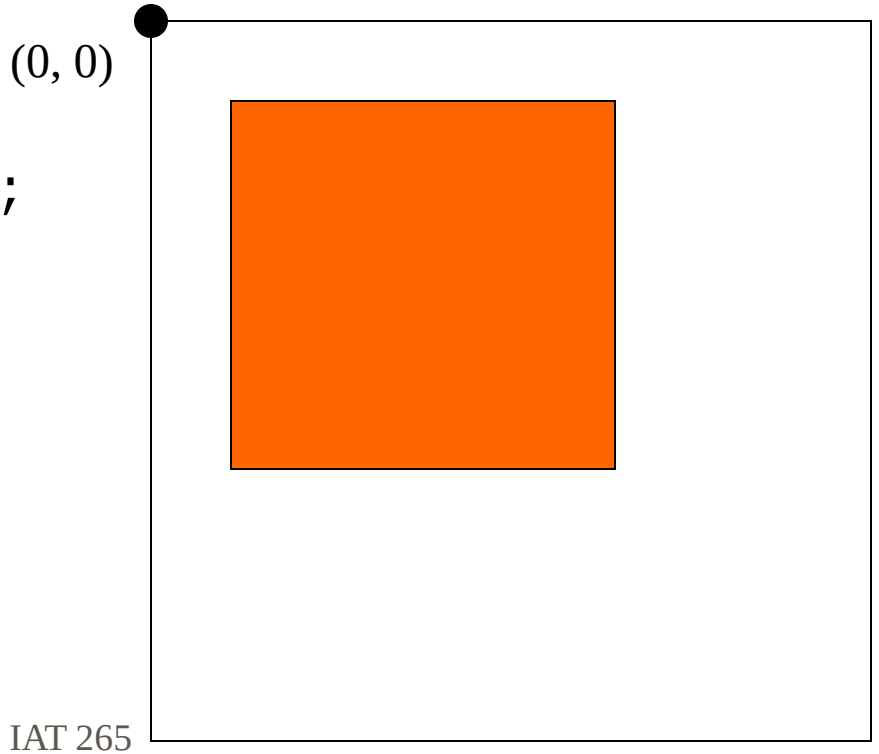


Rotation

- Make a variable with the value for 45 degrees in Radians.

```
float angle = radians(45);
```

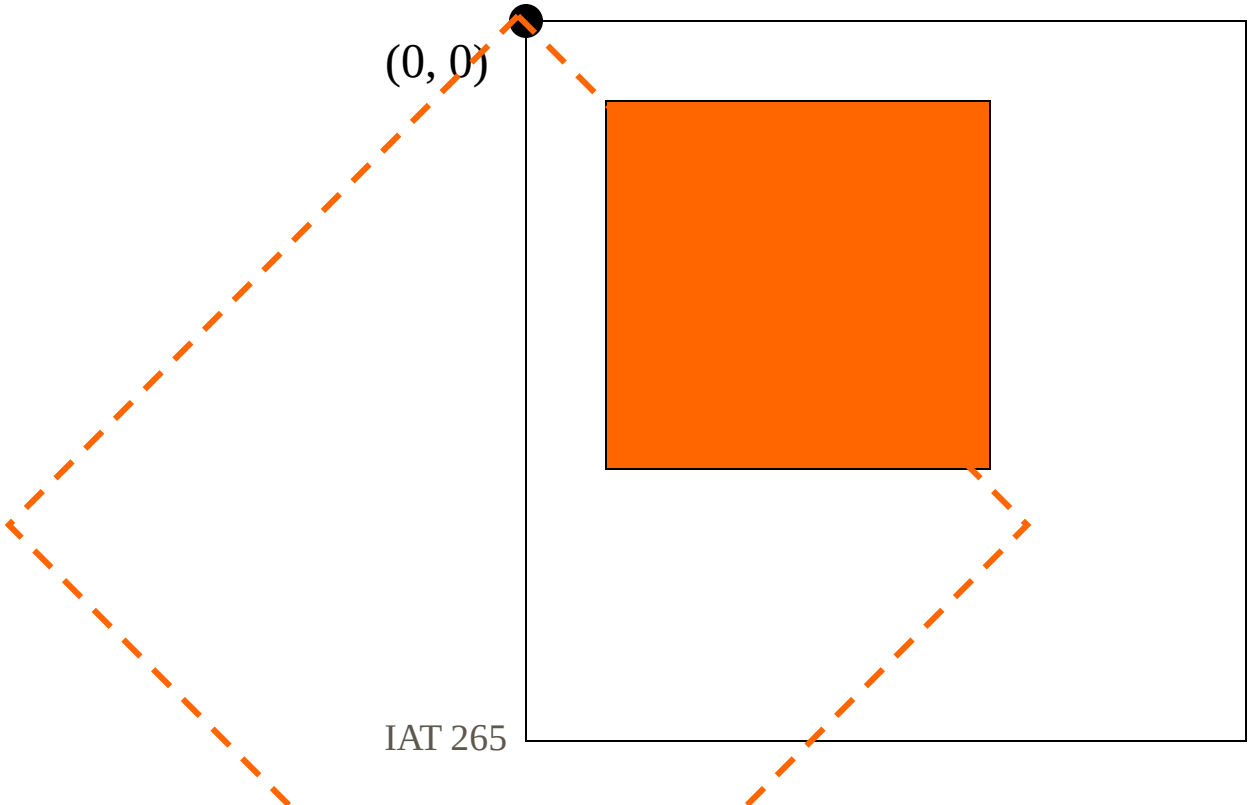
radians() takes an *int* or *float* degree value and returns a *float* radian value



Rotation

- Rotate our drawing canvas 45 degrees around the origin

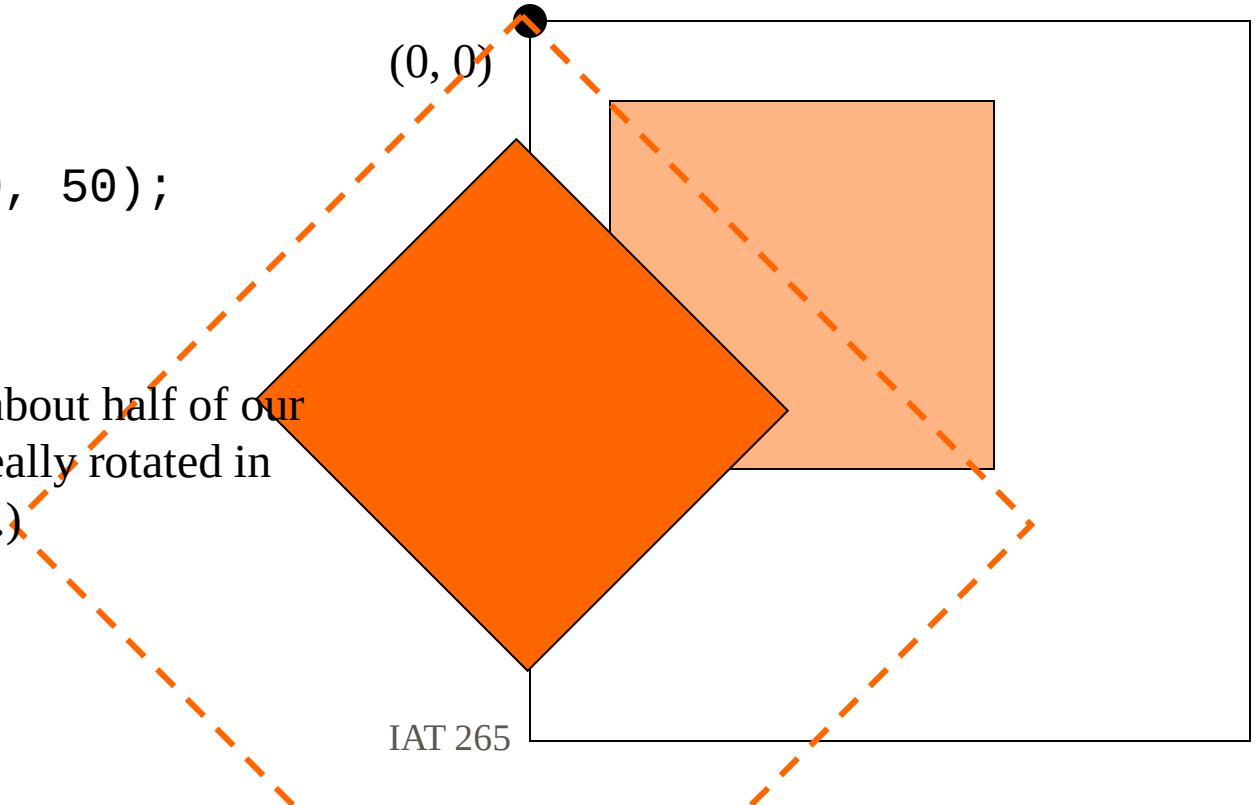
```
rotate(angle);
```



Rotation

- Draw the same square, now on our rotated coordinate system

```
rect(10, 10, 50, 50);
```



(We only get to see about half of our square, and it isn't really rotated in any satisfactory way.)

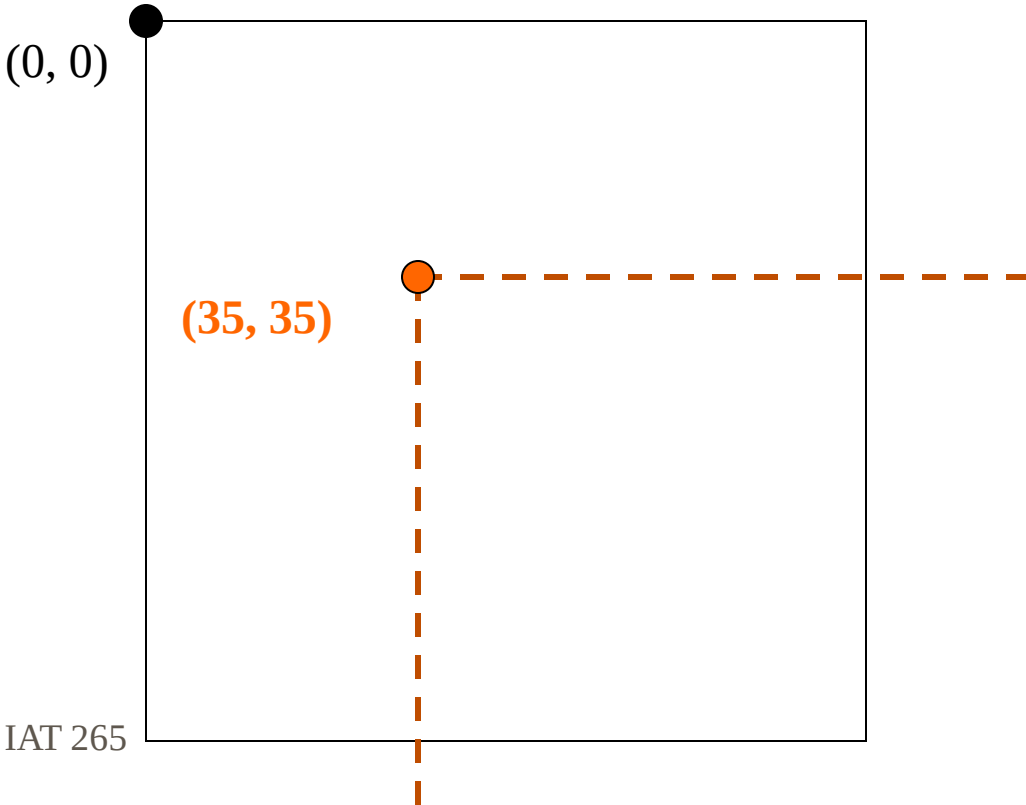
Rotation

- Let's try this from the start, using translation
- Where should we translate to?
  - The point **around** which we want to rotate. So let's try and rotate around the center of the square.
  - This means moving the origin, and drawing the square around it.

Rotation

- Let's start with setting our rotation point:

```
translate(35, 35);
```



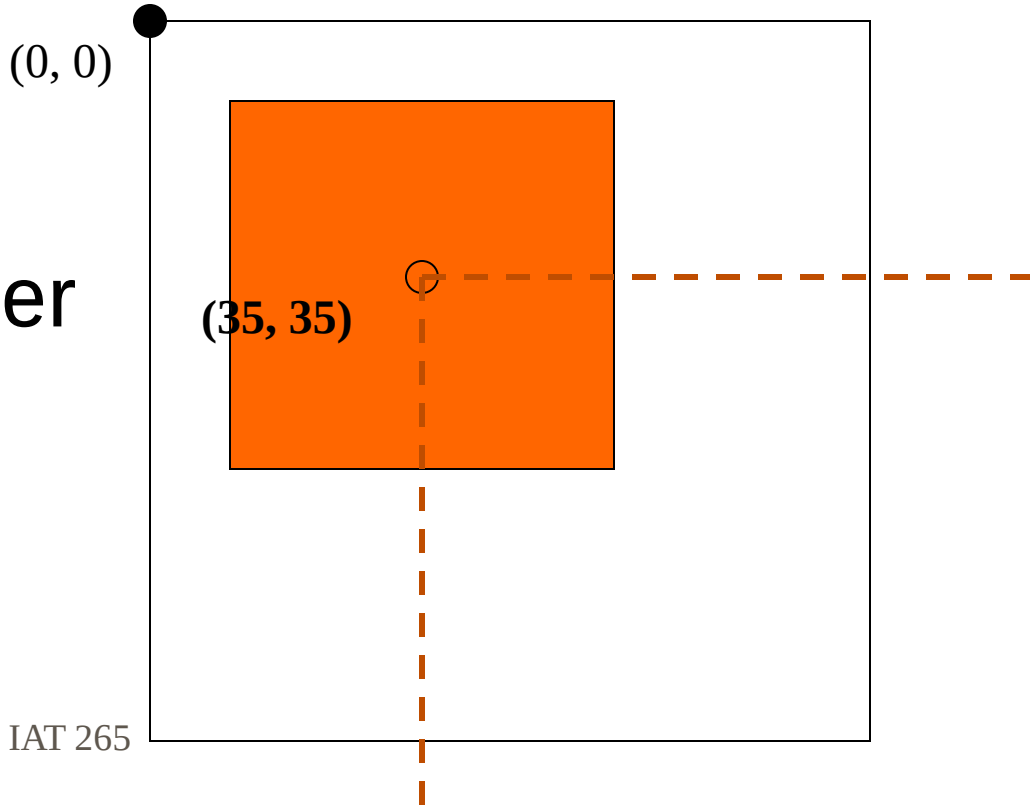
Rotation

- Now let's draw a square with this point at its center.

```
rect( -25, -25, 50, 50);
```

- Or to make it clearer

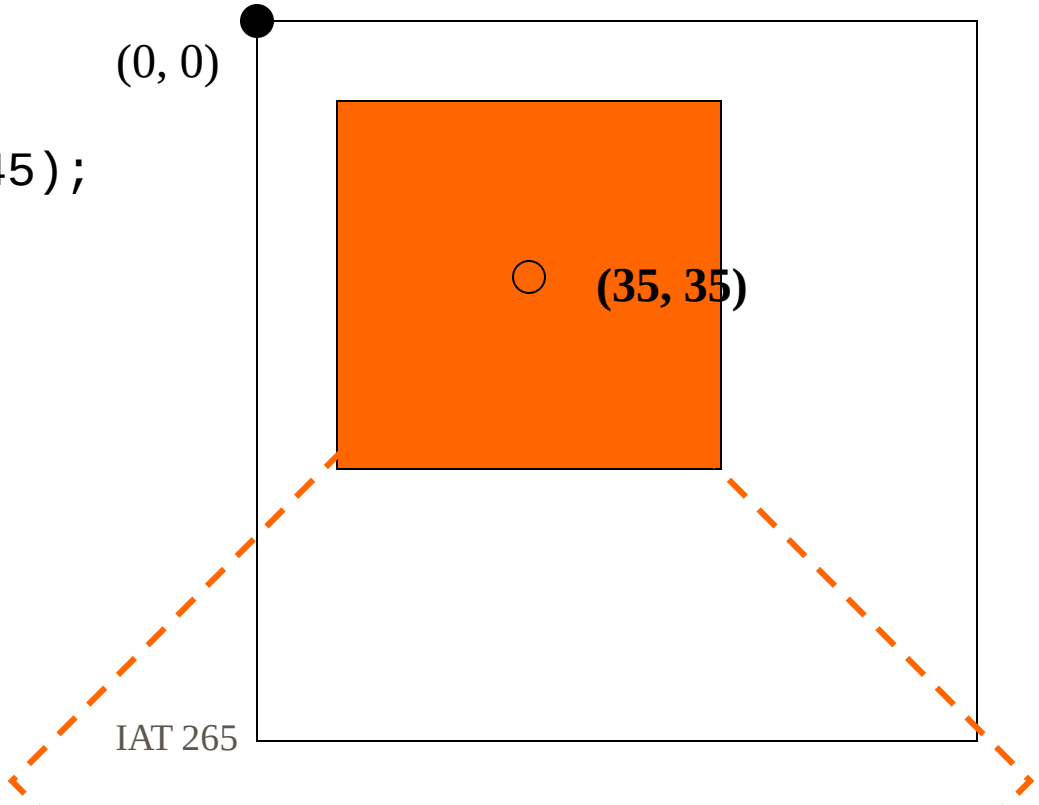
```
rectMode(CENTER);
rect(0, 0, 50, 50);
```



Rotation

- Then let's do the same rotate we did last time

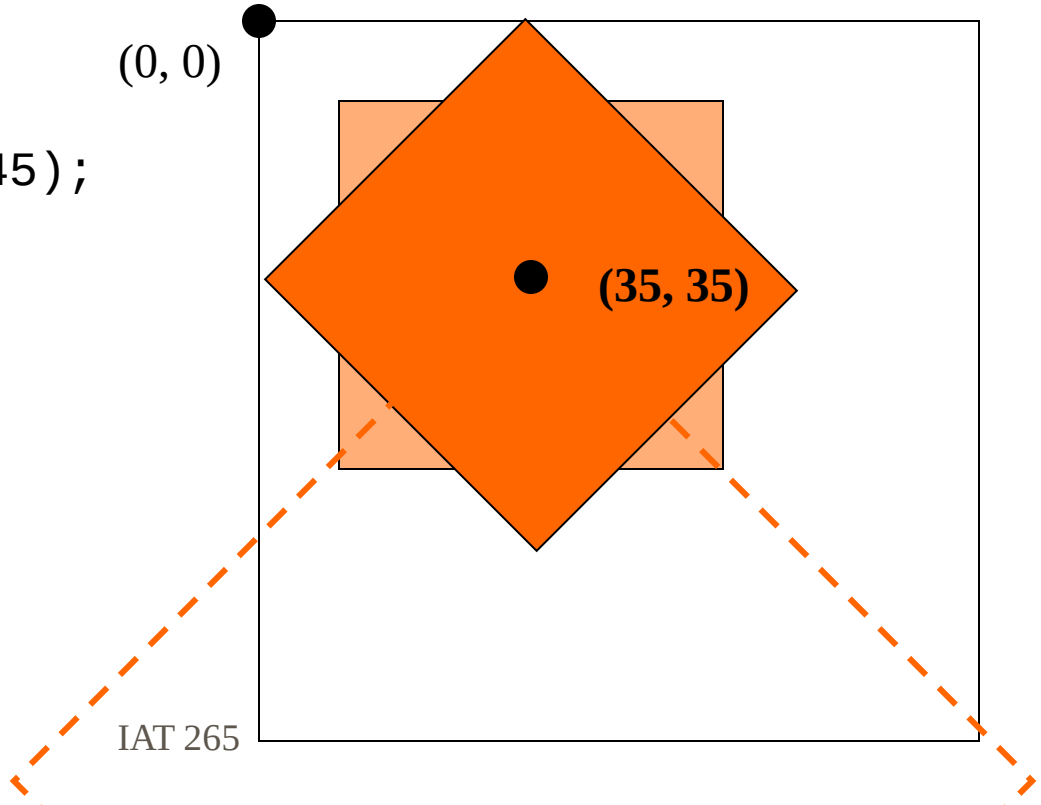
```
float angle = radians(45);
rotate(angle);
```



Rotation

- Now when we draw the same square as before, it will have the same center point.

```
float angle = radians(45);
rotate(angle);
```

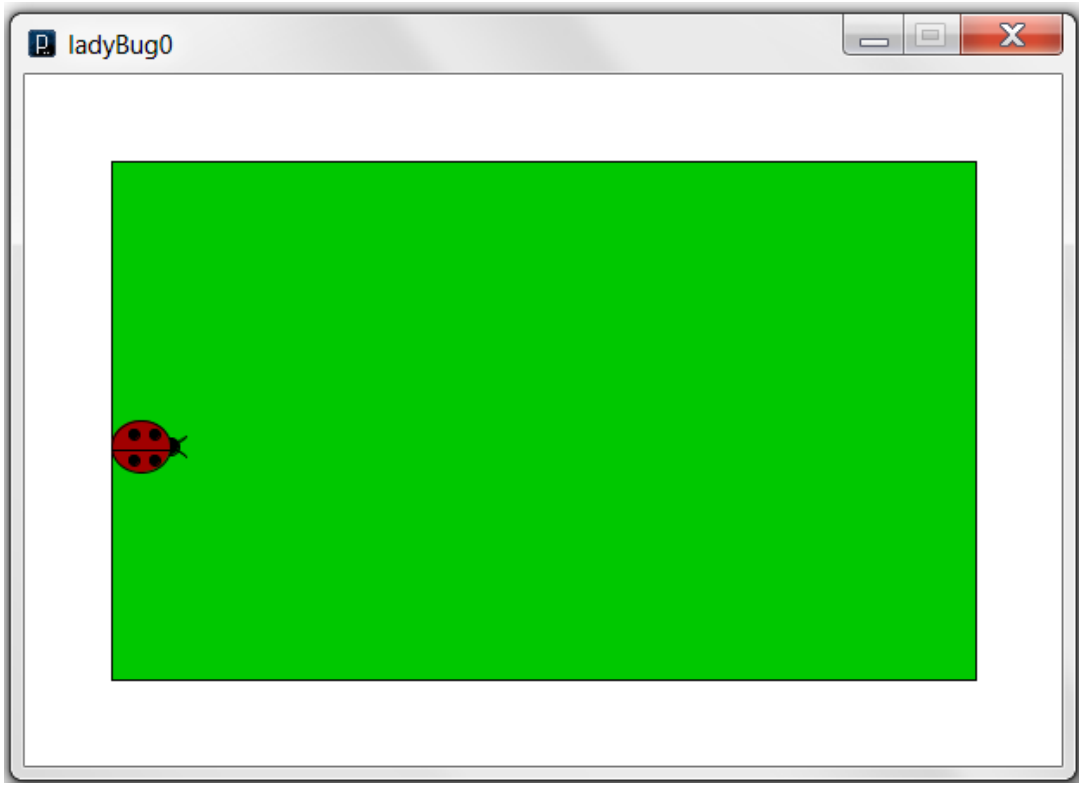


Rotation

- Try applying animations to your rotation using `draw()`
  - Think about: What variable will you want to iterate on to make a shape rotate over time?

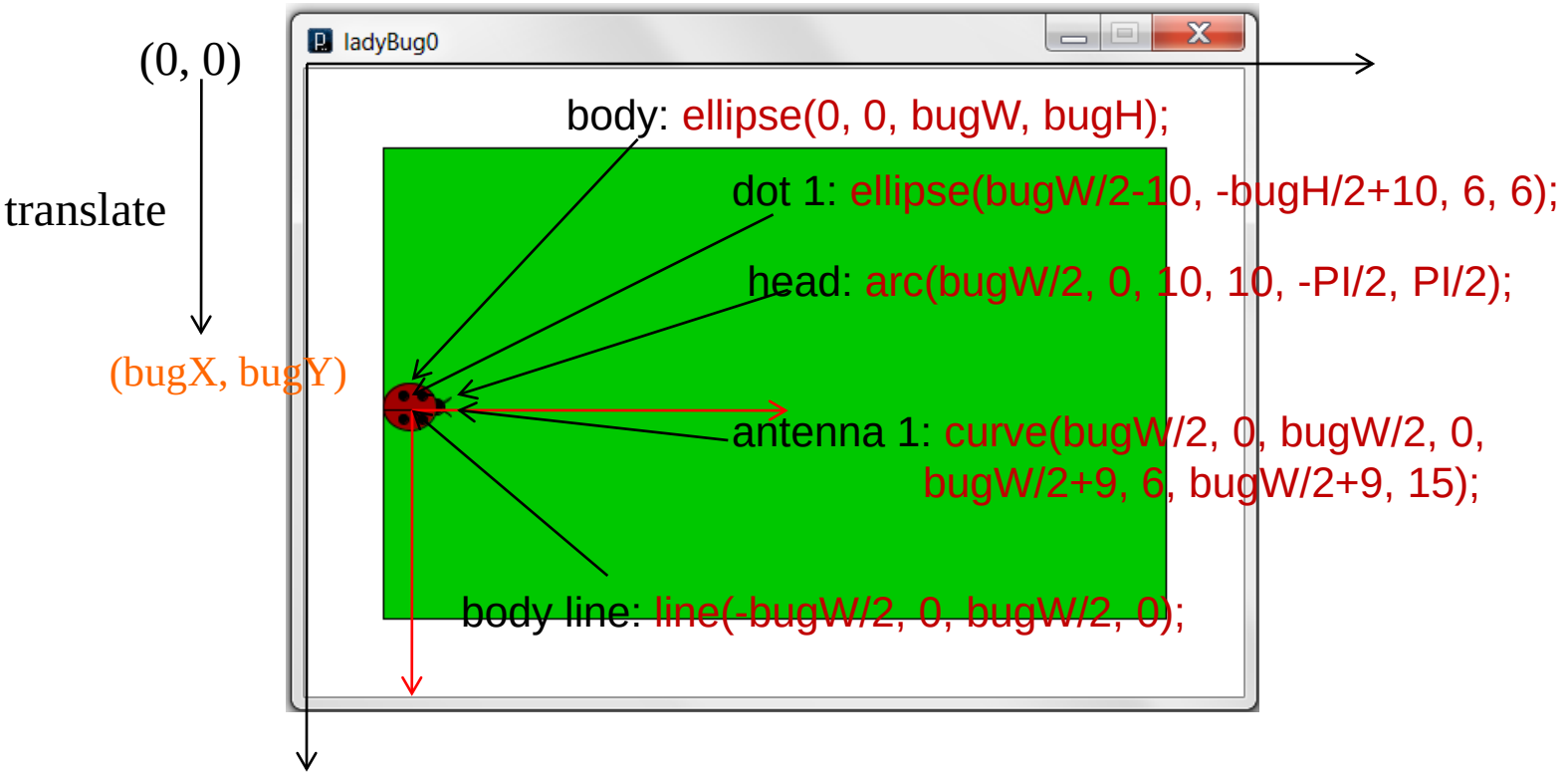
Case study: translate & rotate a ladybug

- Step 1: Draw a Ladybug and place it at middle of the garden's left side



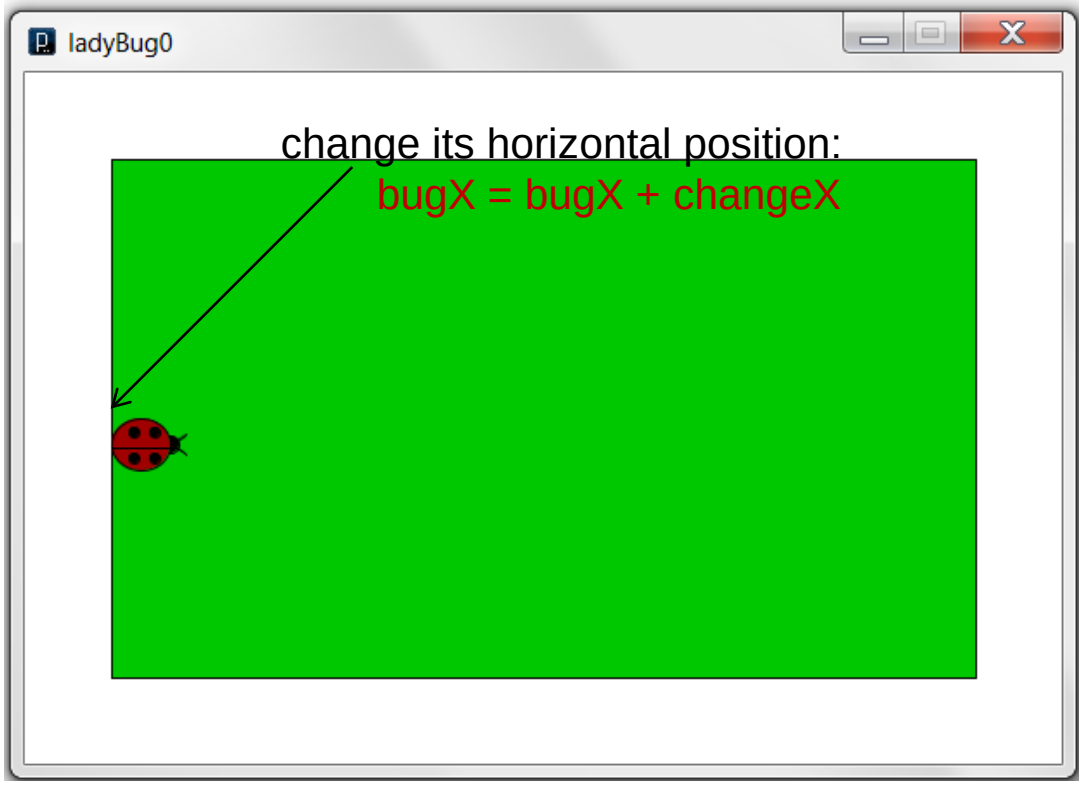
Case study: translate & rotate a ladybug

- Firstly, translate to the upper-left corner of the ladybug, and specify coordinates from it



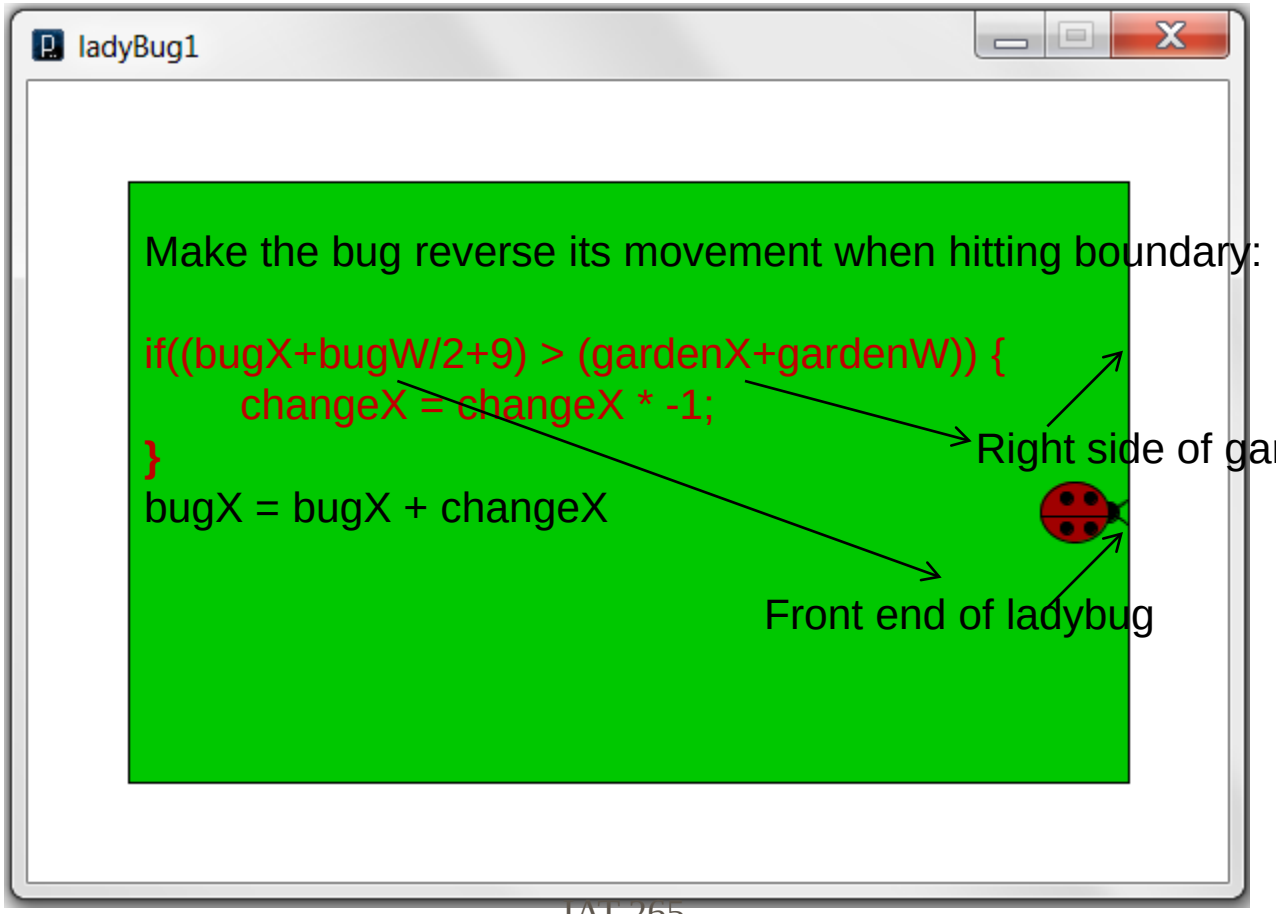
Case study: translate & rotate a ladybug

- Step 2: Move the Ladybug horizontally within the garden



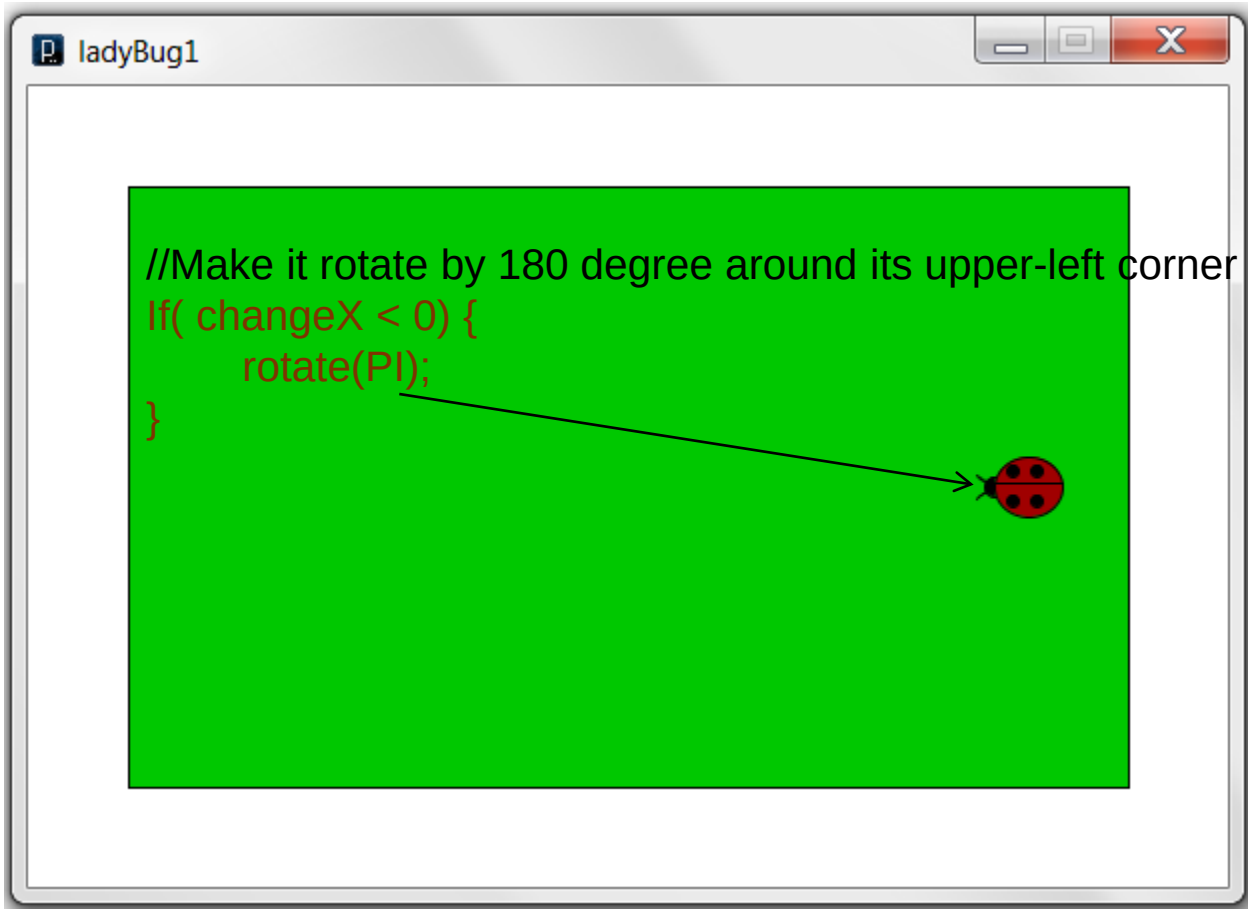
Case study: translate & rotate a ladybug

- Step 3: boundary detection



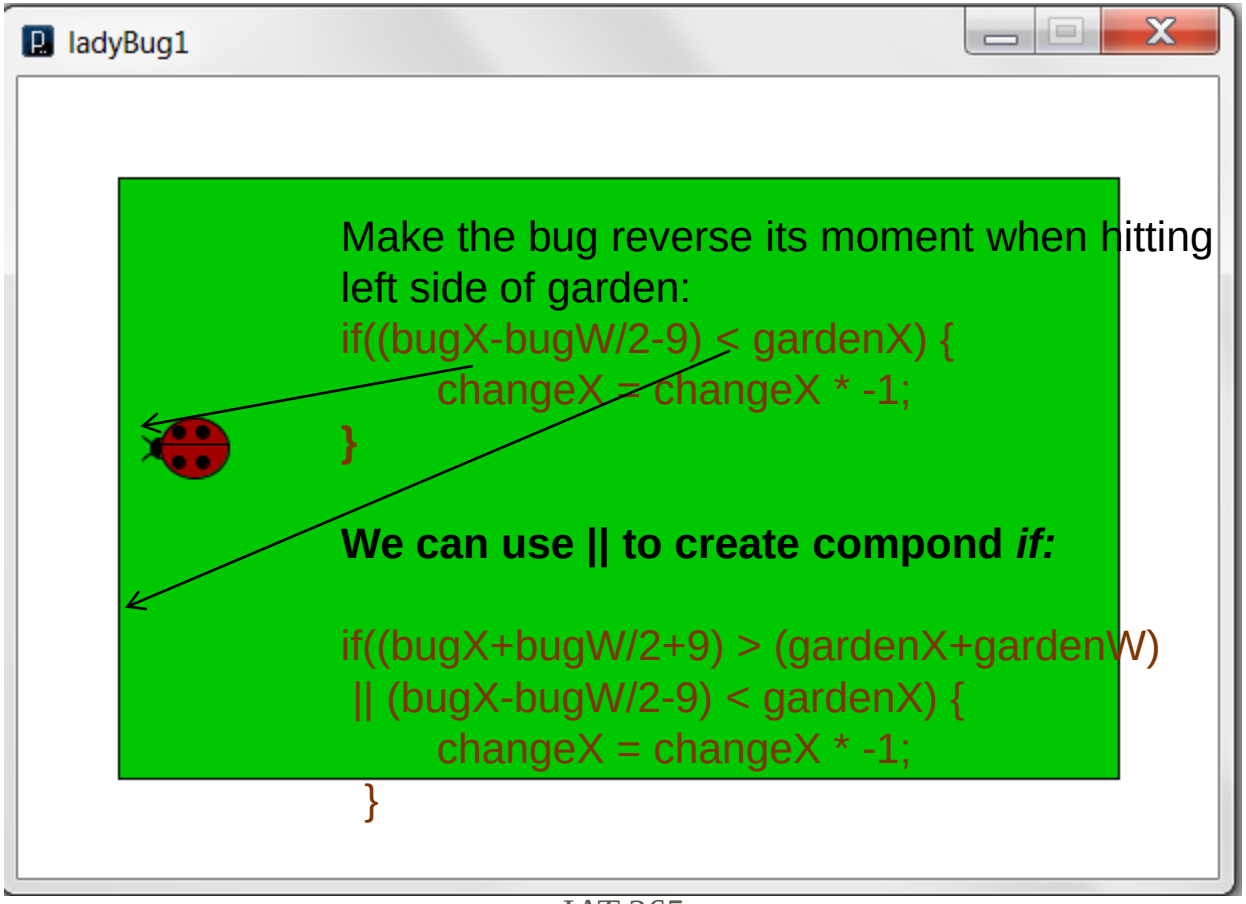
Case study: translate & rotate a ladybug

- Step 4: make the bug rotate when reversing



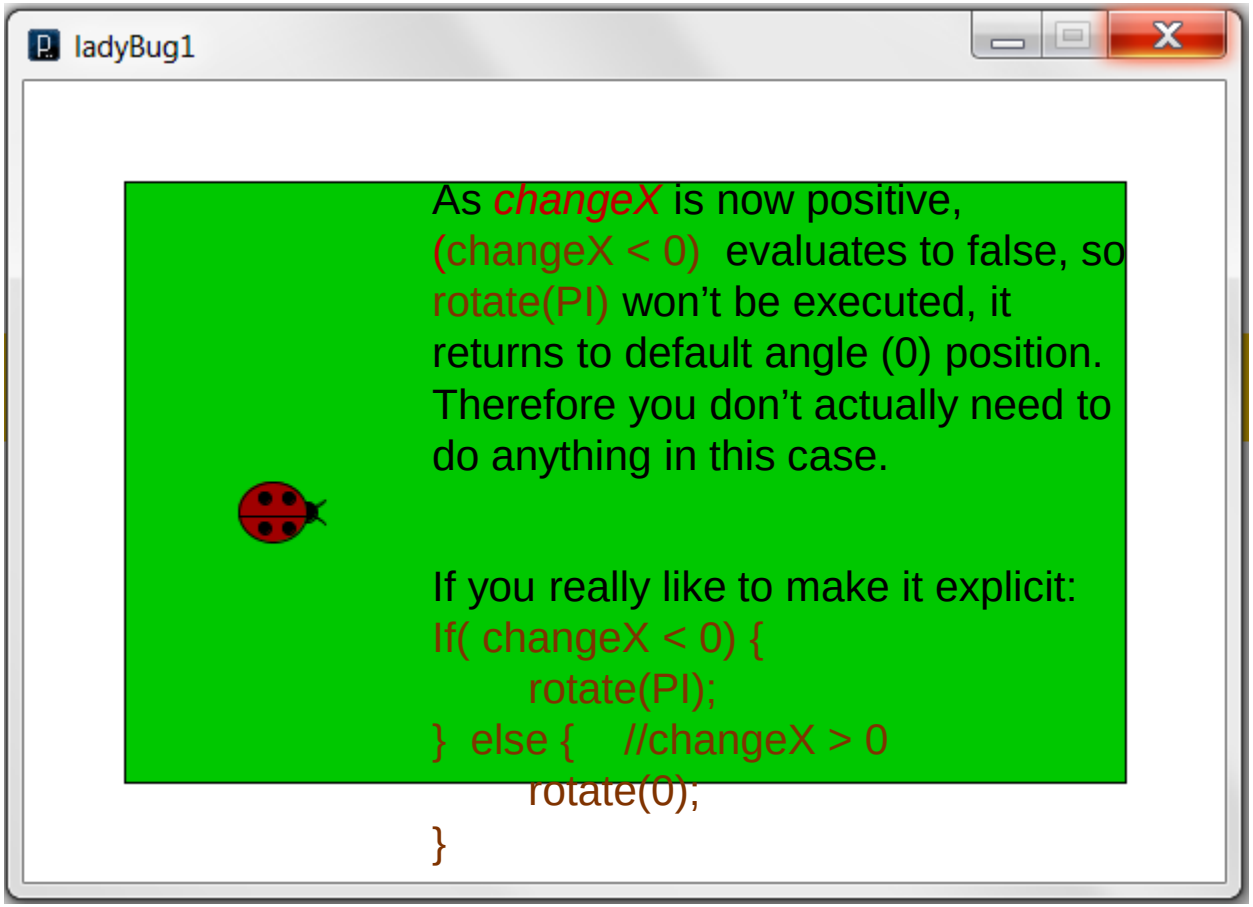
Case study: translate & rotate a ladybug

- Step 5: left boundary detection



Last step : rotate again

- When it hits the left side of the garden



Summary

- Conditionals
  - *If*-statement, *if-else*, *nested if-else*
  - *Compound if*-statement
- Translation and Rotation
- Case study: boundary detection, translate & rotate a ladybug