Review
- Only one statement executes at time
- Scope
  - Global
  - Function
  - Block
- Variable Access Rules
  - Outer scope variables can be accessed from an inner scope
  - Inner scope variables cannot be accessed from an outer scope
- Lifetime
  - Variables come into existence at declaration
  - Variables go out of existence when the block exits
- Shadowing
  - An inner scope variable with the same name as an outer scope variable, shadows (or hides) the outer scope variable from the inner scope.
  - Nevertheless, both variables exist, and are distinct.

Definition
- \( \sin(\Theta) = \frac{o}{h} \)
- \( o = h \sin(\Theta) \)
- \( \cos(\Theta) = \frac{a}{h} \)
- \( a = h \cos(\Theta) \)
- \( \tan(\Theta) = \frac{\sin(\Theta)}{\cos(\Theta)} = \frac{o}{a} = \text{sohcahtoa} \)
**Drawing points along a circle**

```java
int steps = 8;
int radius = 20;
float angle = 2*PI/steps;

for (int i=0; i<steps; i++) {
    float x = sin(angle*i)*radius;
    float y = cos(angle*i)*radius;
    // draw a point every 1/8th of a circle
    ellipse(x, y, 10, 10);
}
```

---

**Up until now ...**

- All movement and sizing of graphical objects have been accomplished by modifying object coordinate values \((x, y)\) and drawing in the default coordinate system.

There is another option...

- We can leave coordinate values unchanged, and modify the coordinate system in which we draw.

---

Three ways to transform the coordinate system:

1. **Translate**
   - Move axes left, right, up, down ...
2. **Scale**
   - Magnify, zoom in, zoom out ...
3. **Rotate**
   - Tilt clockwise, tilt counter-clockwise ...

---

The commands that draw these two ellipses are identical. What has changed is the coordinate system in which they are drawn.

The commands that draw these two ellipses are identical. What has changed is the coordinate system in which they are drawn.
Scale

- All coordinates are multiplied by an x-scale-factor and a y-scale-factor.
- The size of everything is magnified about the origin (0,0)
- Stroke thickness is also scaled.

```plaintext
scale( factor );  // All coordinates are multiplied by the scale factor
scale( x-factor, y-factor );  // Magnifies about the origin
```

The best way to see what is happening, is to look at a grid drawn in the coordinate system.

```plaintext
void grid() {  // Draw grid
  grid(<100, 100, 10, -100, 100, 10>);  // Draw axes
  line(x1, y1, x2, y2);  // Draw grid
  // Draw axes
  float inc = 0.005*width;
  float inc2 = 2.0*inc;
  float inc3 = 4.0*inc;
  fill(0);
  line(x1, 0, x2, 0);
  line(0, y1, 0, y2);
  triangle(x2+inc2, 0, x2, inc, x2, -inc);
  text("x", x2+2*inc2, inc2);
  triangle(0, y2+inc2, 0, inc, y2, -inc);
  text("y", inc2, y2+2*inc2);
  // Draw axes
}
```
void setup() {
  size(500, 500);
  background(255);
  smooth();
}
void draw() {
  grid();
  scale(2,2);
  grid();
}

grid1.pde

void draw() {
  grid();
  fill(255);
  ellipse(50, 50, 40, 30);
  scale(2,2);
  grid();
  fill(255);
  ellipse(50, 50, 40, 30);
}
grid1.pde

void draw() {
  grid();
  translate(250,250);
  grid();
  fill(255);
  ellipse(50, 50, 40, 30);
}
grid2.pde

Transformations can be combined

- Combine Scale and Translate to create a coordinate system with the y-axis that increases in the upward direction
- Axes can be flipped using negative scale factors
- Order in which transforms are applied matters!

Translate

- The origin of the coordinate system (0,0) is shifted by the given amount in the x and y directions.

  `translate(x-shift, y-shift);`
void draw() {
    translate(0, height);
    scale(4, -4);
    grid();
}

grid3.pde

Rotate
— The coordinate system is rotated around the origin by the given angle (in radians).

rotate( radians );

grid4.pde

void draw() {
    translate(250.0, 250.0);
    rotate( 25.0 * (PI/180.0) );
    //scale( 2 );
    grid();
}

grid4.pde

void draw() {
    translate(250.0, 250.0);
    rotate( 25.0 * (PI/180.0) );
    scale( 2 );
    grid();
}

grid4.pde

void draw() {
    translate(250.0, 250.0);
    rotate( 25.0 * (PI/180.0) );
    scale( 2 );
    grid();
}

grid4.pde
void draw() {
    grid();
    fill(255);
    ellipse(50, 50, 40, 30);
    translate(250.0, 250.0);
    rotate(25.0 * (PI/180.0));
    scale(2);
    grid();
    fill(255);
    ellipse(50, 50, 40, 30);
}

Some things to remember:
1. Transformations are cumulative.
2. All transformations are cancelled each time draw() exits.
   - They must be reset each time at the beginning of draw() before any drawing.
3. Rotation angles are measured in radians
   - \( \pi \) radians = 180°
   - radians = (PI/180.0) * degrees
4. Order matters

---

void setup() {
    size(500, 500);
    smooth();
    noLoop();
}

void draw() {
    background(255);
    fill(0);
    translate( width/2, height/2 );
    for (int i=0; i<36; i++)
    {
        text( i, 0.0, -150.0 );
        rotate(10.0 * (PI/180.0));
    }
}

// example3.pde
Each time through the loop an additional 10 degrees is added to the rotation angle.
Total rotation accumulates.

float start = 0.0;

void setup() {
    size(500, 500);
    smooth();
}

void draw() {
    background(255);
    fill(0);
    translate( width/2, height/2 );
    rotate(start);
    for (int i=0; i<36; i++)
    {
        text( i, 0.0, -150.0 );
        rotate(10.0 * (PI/180.0));
    }
    start += 1.0*(PI/180.0) % TWO_PI;
}

// example4.pde
Each time through the loop an initial rotation angle is set, incremented, and saved in a global.
Transformations reset each time draw() is called.