Variables

When you declare a variable, first write the **DATA TYPE**, then the **NAME OF THE VARIABLE** and then you can either leave the value unspecified (and define it later) or define it there on the spot by adding an equal signs and assigning it a value.

For example. If you know you only need whole numbers for a value, an integer (or int) is most appropriate:

int - whole numbers, positive or negative. *See More on reference page.*
int age = 26;

float - floating point number or decimal, positive or negative. *See More on reference page.*
float sensorValue = 2.751;

String - a series of characters (ie, text) as opposed to numbers. *See More on reference page.*
String currentName= "Grizelda";

boolean - true or false (1 = true, 0 = false). *See More on reference page.*
boolean mode1 = false;

char - a single character such as a letter or a symbol. *See More on reference page.*
char letter = 'A';

Variables

Variables are used to store values. When we declare a variable, we first tell Processing what type of information that variable will store. Think how a program would need treat a name such as "Sarah" (called a STRING datatype in processing) differently from numeric information such as a temperature reading like 98.6 (called a FLOAT data type in processing).

When you name a variable, remember that the name is Case-sensative and that the the name must not begin with special characters.

```
String name = "ham";  // Declare and assign string variable
int counter = 12;  // Declare and assign integer
counter = 32;  // Declare and assign a value to 'counter'
print(counter);
println(counter); //  println prints the line with a carriage return afterwards
```

```
int x;          // Declare the variable x of type int
float y;        // Declare the variable f of type float
boolean b;      // Declare the variable b of type boolean
x = 50;         // Assign the value 50 to x
y = 12.6;       // Assign the value 12.6 to f
b = true;       // Assign the value true to b
```

Scope of Variables

Local Variables

Declared inside a function.
Can only be used inside the function where it is declared.

```
void setup(){
      int bgColor = 200;    // local variable
      background(bgColor); // use local variable
}

void draw(){
      background(bgColor);
      line(0, 0, width, height);
}
```

← Error! cannot use bgColor here without declaring it as a local variable (again)

Scope of Variables

Global Variable

Declared outside the setup() and draw().
Can be used anywhere in your sketch.

```
int bgColor = 200; // global variable

void setup(){
   background(bgColor); // use global variable
}

void draw(){
   background(bgColor); //use global variable
   line(0, 0, width, height);
}
```

Scope of Variables

If a local variable is declared with the same name as a global variable, the program will use the local variable to make its calculations within the function it is sitting in.

```
int bgColor = 200;   // set global variable

void setup() {
  int bgColor = 100;// local variable, redefine the
                        value of bgColor
background(bgColor); // uses local variable value
}

void draw() {
  background(bgColor); // use global variable because
                          we are in a different function
  line(0, 0, width, height);
}
```

Conditionals and *Program Flow* *(link)*

https://medium.com/@AlexanderObregon/getting-started-with-java-control-flow-statements-f1cea47e82bd

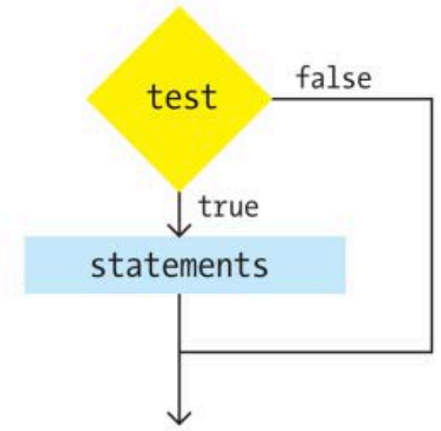
Statements within the if section are only executed in case a condition such as (mouseX < 150) is true; statements within the else section are executed only in case the < condition is false.

```
void setup () {
  size(300, 300);
}

void draw() {
  if(mouseX < 150) {
    line( 0, mouseY, 150, mouseY );
  }
  else {
    line( 150, mouseY, 300, mouseY );
  }
}
```

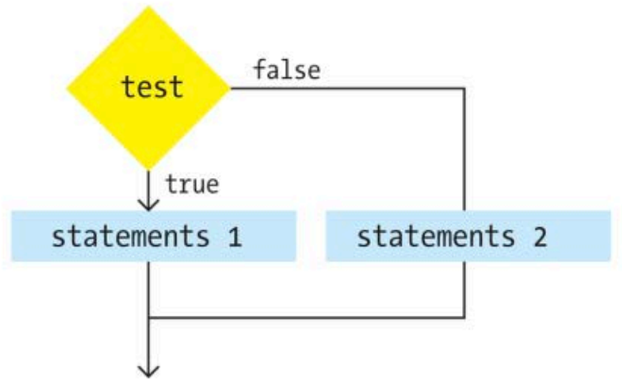
Conditionals **and** *Program Flow* *(link)*

LAYOUTS

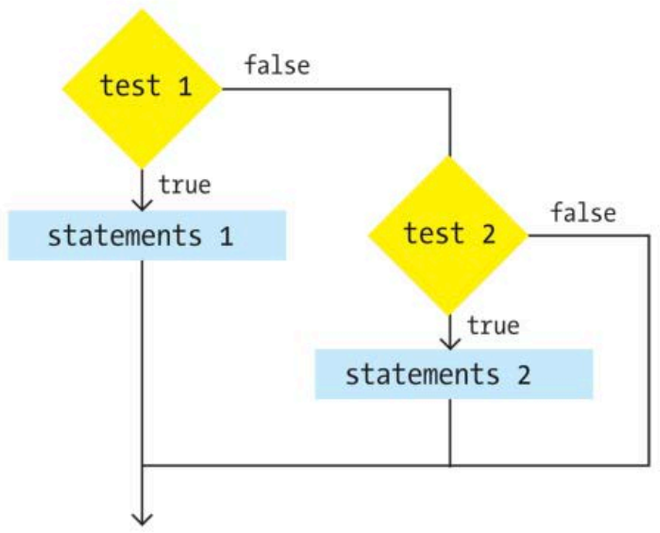
```
if (test) {
  statements
}
```




```
if (test) {
  statements 1
} else {
  statements 2
}
```



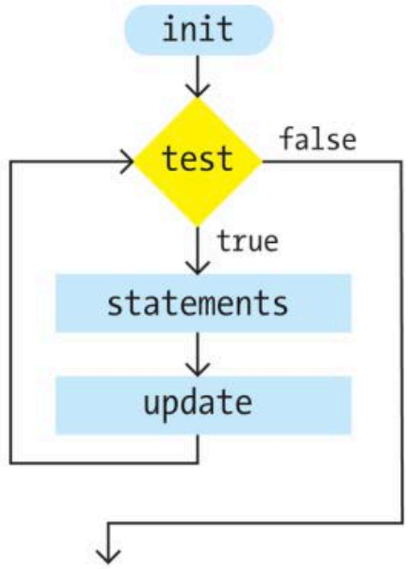

```
if (test 1) {
  statements 1
} else if (test 2) {
  statements 2
}
```



Loops: **For and While**

The **`for()`** loop uses defined conditions

```
for (int i=40; i<80; i=i+5) {
  line(30, i, 80, i);
}
```

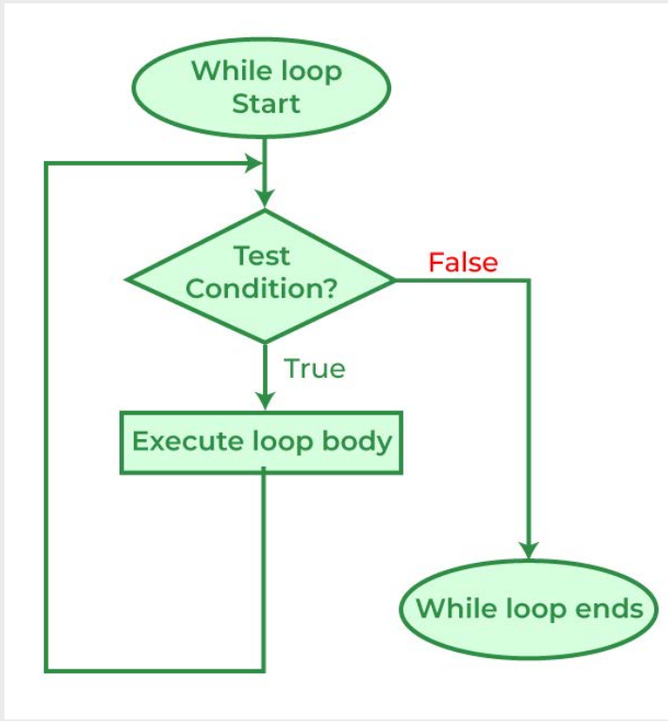



```
for (init; test; update) {
  statements
}
```

The **`while()`** loop repeats as long as the condition is true

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

note: if the test condition in the while loop cannot be false, the program freezes



https://processing.org/reference#math-operators

Relational Operators

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

```
5 > 4   // True
5 < 3   // False
5 > 5   // False
5 >= 5  // True
5 >= 6  // False
5 != 6  // False(not equal)
5 == 5  // True
5 == 4  // False
```

```
++  (x++ is shorthand for x = x + 1)
+=  (increment up; x += 5 means add 5 with each loop)
-=  (decrement down; x -= 5 means subtract 5 with each loop)
*=  (multiply by)
/=  (divide by)
```

Logic Operators

Compound logic uses more than one conditional argument and combines those arguments into logical operators.

```
a || b (OR)  // if either a or b is true then ...
a && b (AND) // if both a AND b are true then ...
!a (NOT) // if a is false then...
```

Mouse Interaction

mouseX, mouseY
: Stores the current position of the mouse inside the window

pmouseX, pmouseY
: Stores the position of the mouse in the previous frame inside the window

mousePressed, mousePressed()
: Used to detect if/when the mouse is being pressed (clicked)

mouseButton
: Stores information about what button is being pressed

mouseReleased()
: Called every time the mouse is released

mouseDragged()
: Called every time when the mouse is dragged (pressed and moved)

mouseMoved()
: Called every time when the mouse moves and not pressed

Mouse Interaction

mouseX, mouseY

```
void setup()
{
   size(500, 200);
   strokeWeight(5);
   stroke(0, 100);
   smooth();
}

void draw()
{
   ellipse(mouseX, mouseY, 5, 5);
}
```

Mouse position

`mouseX`: X-Position the mouse

`mouseY`: Y-Position

```
line(mouseX, 20, mouseX, 80);
```

mousePressed returns true while mouse is pressed, false if not.

```
void draw() {
  if (mousePressed == true) {
    fill(0);
  } else {
    fill(255);
  }
  rect(25, 25, 50, 50);
}
```

Mouse Interaction I

mouseX, mouseY

pmouseX, pmouseY

pmouseX and pmouseY contains the previous horizontal and previous vertical coordinate of the mouse. It is the position of the mouse in the frame previous to the current frame.

This is very useful to determine the velocity of a mouse movement or gesture. By subtracting the previous from the current mouse position the current mouse velocity can be determined.

```
void draw()
{
   background(204);
   line(mouseX, 20, pmouseX, 80);
}
```

Mouse Interaction II

mousePressed

mousePressed is a system variable which is true if the button is pressed and false if the button is not pressed.

```
void setup() {
   size(200, 200);
   rectMode(CENTER);
   background(255);
   smooth();
   noStroke();
}

void draw() {
   if(mousePressed == true) {
      fill(random(255), 100);
   } else {
      fill(0);
   }
   rect(mouseX, mouseY, 30, 30);
}
```

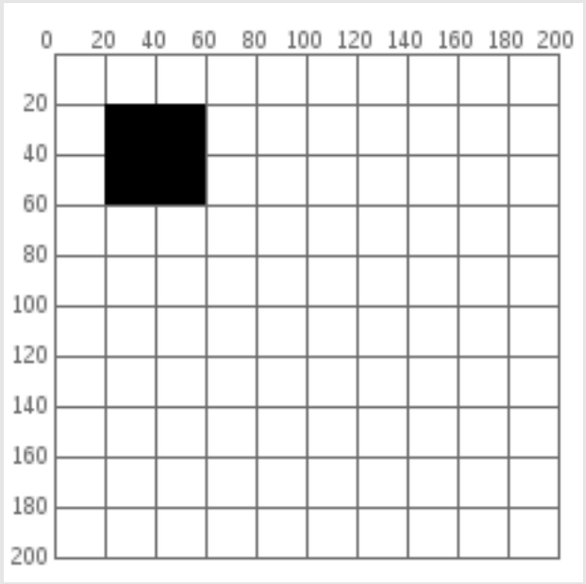
Mouse Interaction III

mousePressed()

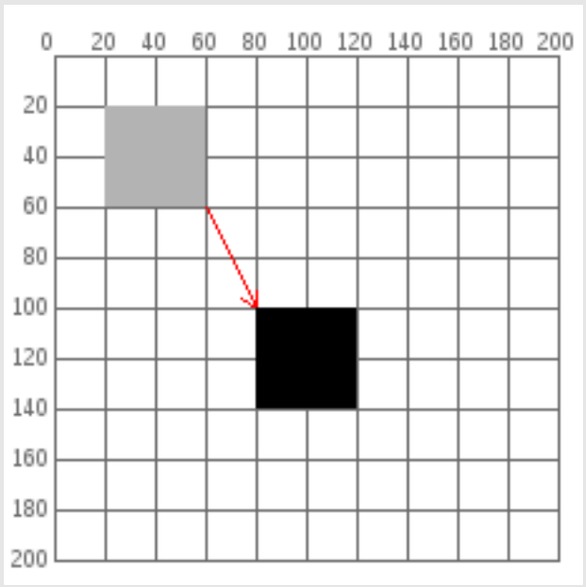
The mousePressed() system function is called every time the mouse button is pressed.

```
int fillColor = 0;
void draw() {
   fill(fillColor);
   rect(25, 25, 50, 50);
}
void mousePressed()
{
   if(fillColor == 0) {
      fillColor = 255;
   } else {
      fillColor = 0;
   }
}
```

Transformations

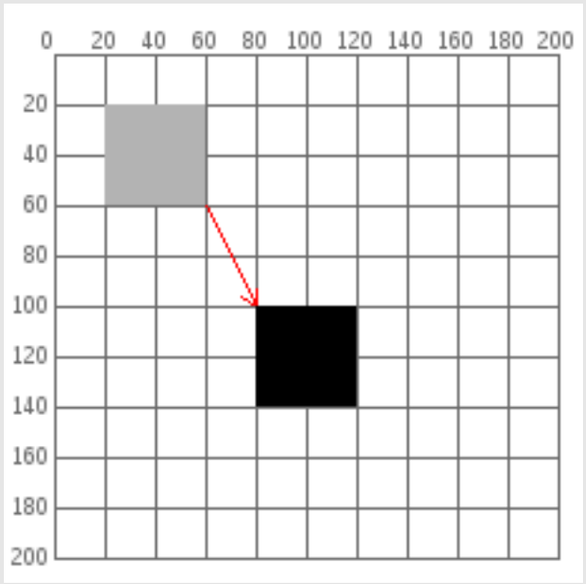



```
rect(20, 20, 40, 40)
```



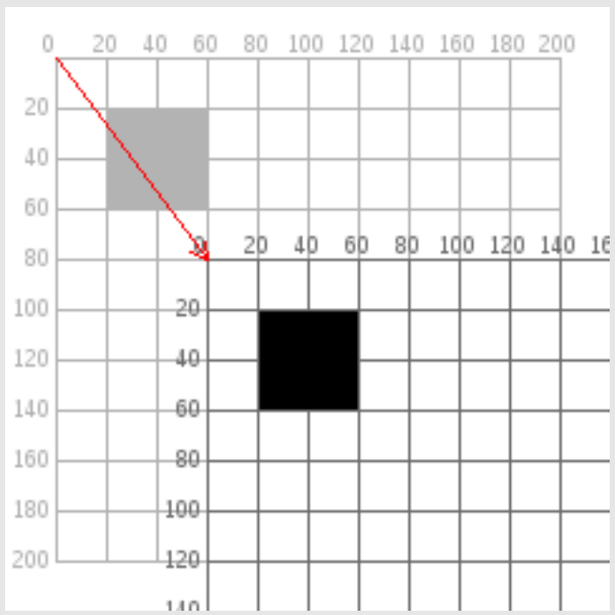

```
rect(20 + 60, 20 + 80, 40, 40)
```

Translate




```
rect(20 + 60, 20 + 80, 40, 40)
```

Using transformations: we move the coordinate system instead of individual objects




```
translate(60, 80);
rect(20, 20, 40, 40)
```

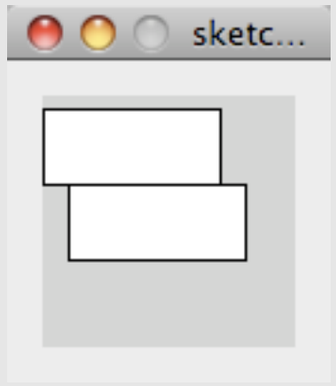
Translate

Follow the mouse using translate.

```
void setup()
{
  size(200, 200);
  noStroke();
  fill(255, 0, 0);
}

void draw()
{
  background(255);

  translate(mouseX, mouseY);
  ellipse(0, 0, 40, 40);
}
```

Translate

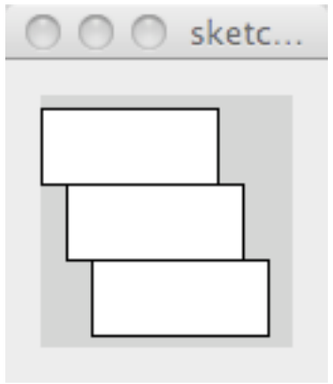
Only the second rect is affected by translate




```
rect(0, 5, 70, 30);
translate(10, 30);
rect(0, 5, 70, 30);
```

Translate

Transformation accumulate (are additive)




```
rect(0, 5, 70, 30);
translate(10, 30);
rect(0, 5, 70, 30);
translate(10, 30);
rect(0, 5, 70, 30);
```

Translate

```
void setup()
{
  size(200, 200);
  noStroke();
}

void draw()
{
  background(255);

  fill(255, 0, 0, 100);
  translate(mouseX, mouseY);
  ellipse(0, 0, 40, 40);

  // Note that this ellipse will move twice as fast as the previous,
  // because transformation accumulate
  fill(0, 255, 0, 100);
  translate(mouseX, mouseY);
  ellipse(0, 0, 40, 40);

}
```

Translate

```
void setup()
{
  size(200, 200);
  background(255);
  noStroke();

  // draw the original position in gray
  fill(192);
  rect(20, 20, 40, 40);

  // draw a translucent red rectangle by changing the coordinates
  // passed to the rect function
  fill(255, 0, 0, 128);
  rect(20 + 60, 20 + 80, 40, 40);

  // draw a translucent blue rectangle by translating the grid
  fill(0, 0, 255, 128);
  translate(60, 80);
  rect(20, 20, 40, 40);
}
```

Rotate

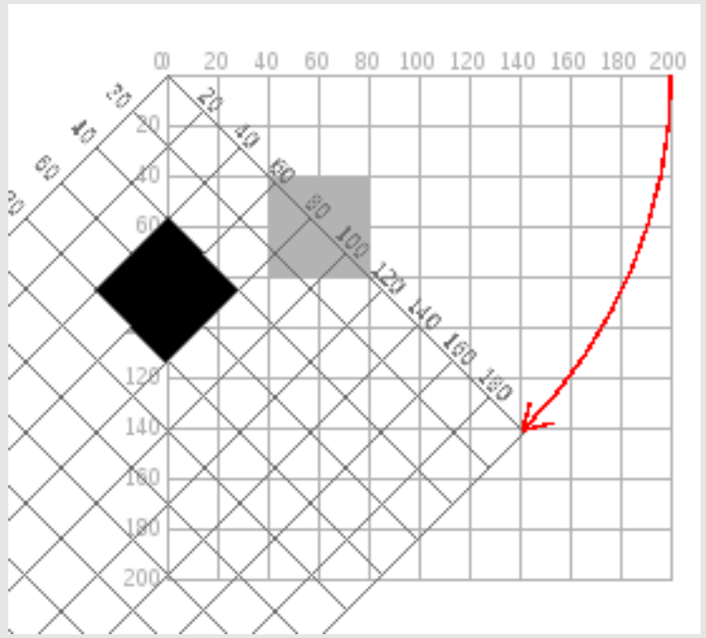
The rotate function rotates the coordinate system allowing you to draw shapes at an angle

Rotations are specified in radians and in clockwise direction.

Rotations are also accumulated.

You can transform values to/from radians/degrees using the functions

```
radians(float value);
degrees(float value);
```




```
smooth();
rect(55, 0, 30, 45);
rotate(PI/8);
rect(55, 0, 30, 45);
```

What are radians? See:
https://study.com/learn/lesson/radian-measure-formula-angle.html

Rotate

To rotate objects from their center, you need to combine translation + rotation.

Example with a square:

A. Translate the coordinate system's origin (0, 0) to where you want the upper left of the square to be.
B. Rotate the grid PI/4 radians (45°)
C. Draw the square at the origin.

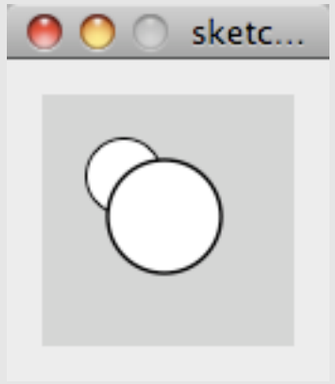
```
// Draw from the center
rectMode(CENTER);

// move the origin to the pivot point
translate(width/2, height/2);

// then pivot the grid
rotate(radians(45));

// and draw the square at the origin
fill(0);
rect(0, 0, 40, 40);
```

Scale

The scale function scales the coordinate system allowing you to draw shapes at an different sizes

The scales are specified as percentages in decimal mode: 2.0 = 200%.

Be careful, as scaling also affects the position of objects that are not drawn at 0, 0.

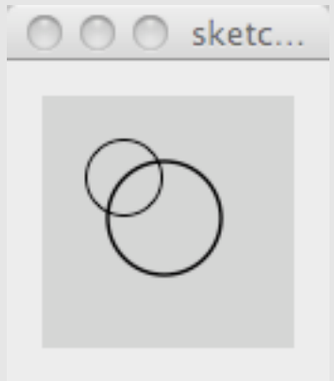



```
smooth();
ellipse(32, 32, 30, 30);
scale(1.5);
ellipse(32, 32, 30, 30);
```

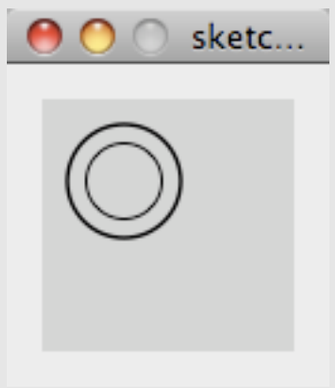
Scale

Be careful, as scaling also affects the position of objects that are not drawn at 0, 0.

See in this example, how drawing the ellipses at (32, 32) vs drawing them at (0, 0) with a translation of (32, 32) makes a different outcome when scale is involved.




```
noFill();
smooth();
ellipse(32, 32, 30, 30);
scale(1.5);
ellipse(32, 32, 30, 30);
```




```
noFill();
smooth();
translate(32, 32);
ellipse(0, 0, 30, 30);
scale(1.5);
ellipse(0, 0, 30, 30);
```

Examples

Drawing a color wheel

```
void setup() {
  size(200, 200);
  background(255);
  smooth();
  noStroke();
}
```

```
void draw(){
  if (frameCount % 10 == 0) {
    fill(frameCount * 3 % 255, frameCount * 5 % 255,frameCount * 7 % 255);

    translate(100, 100);
    rotate(radians(frameCount * 2  % 360));

    rect(0, 0, 80, 20);
  }
}
```

Examples

Using a for loop to accumulate transformations

```
size(200, 200);
background(0);
smooth();
stroke(255, 100);

translate(width/2, 80);
for ( int i = 0; i < 18; i++ )
{
  strokeWeight(i);
  rotate(PI/12);
  line(0, 0, 55, 0);
}
```