Review

- Hue-Saturation-Brightness vs. Red-Green-Blue color
- Decimal, Hex, Binary numbers and colors
- Variables and Data Types
- Other "things," including Strings and Images
- Operators: Mathematical, Relational and Logical
- Expressions and Expression Evaluation (PEMDAS)
- Conditionals: if, if/else, if/else if/else, statements
- Conditionals: switch statement
## Evaluating Logical Expressions

<table>
<thead>
<tr>
<th>Negation ( !A )</th>
<th>A</th>
<th>!A</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>true</td>
<td>true</td>
<td>false</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Conjunction (A &amp;&amp; B)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
</tr>
<tr>
<td>----</td>
</tr>
<tr>
<td>true</td>
</tr>
<tr>
<td>true</td>
</tr>
<tr>
<td>false</td>
</tr>
<tr>
<td>false</td>
</tr>
</tbody>
</table>

| Disjunction (A || B) |
|----------------------|
| A | B | A || B |
|----|---|--------|
| true | true | true |
| true | false | true |
| false | true | true |
| false | false | false |

Derive new tables by combining operators…

1. *If I've already had two desserts, then don't serve me any more. Otherwise, I'll take another, thank you.*

   \[ A = \text{had\_dessert\_1}, \ B = \text{had\_dessert\_2} \]

2. *I'll have dessert, as long as it is not flan (A) or beef jerky (B).*
Conditionals: switch-statement

• Works like an if-statement, only ...
  – Expression returns any value (not limited to a boolean)
  – The first option (case) with an equivalent value is executed.
• Convenient for large numbers of value tests.

```java
switch( expression ) {
    case label1:         // label1 equals expression
        statements;
        break;
    case label2:         // label2 equals expression
        statements;
        break;
    default:             // Nothing matches
        statements;
}
```
Two Ways to Implement the Same Logic Using If/Else & Switch

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

void draw() {}

void keyPressed()
{
    switch(key) {
        case 'a':
        case 'A':
            println("Turning left");
            break;
        case 's':
        case 'S':
            println("Turning right");
            break;
    }
}

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

void draw() {}

void keyPressed()
{
    if (key == 'a' ||
        key == 'A') {
        println("Turning left");
    }
    else if (key == 's' ||
        key == 'S') {
        println("Turning right");
    }
}
int positionX = 250;
int positionY = 250;
int deltaX = 0;
int deltaY = 0;

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

void draw() {
  background(255);
  // Increment position and clip value
  positionX += deltaX;
  positionY += deltaY;

  // Clip values
  if (positionX < 0) positionX = 0;
  if (positionX > width) positionX = width;
  if (positionY < 0) positionY = 0;
  if (positionY > height) positionY = height;

  // Draw ellipse
  ellipse(positionX, positionY, 50, 50);
}

void keyPressed() {
  // Change direction based on key code
  switch (keyCode) {
    case LEFT:
      deltaX = -2;
      deltaY = 0;
      break;
    case RIGHT:
      deltaX = 2;
      deltaY = 0;
      break;
    case UP:
      deltaY = -2;
      deltaX = 0;
      break;
    case DOWN:
      deltaY = 2;
      deltaX = 0;
      break;
    case ENTER:
      deltaX = 0;
      deltaY = 0;
      break;
  }
}

Note the distinction between state (keyPressed) and behavior (draw).
The Walker – Version 2

boolean walkPose = false;   // Current walk pose

float speed = 5.0;          // Max walking
float cX = 100.0;           // Current walker location
float cY = 100.0;

void setup() {
    size(500, 500);
    smooth();
    frameRate(20);
}
void draw() {
    background(255);
    fill(200);
    stroke(0);

    // Draw the walker
    // Head and body
    line(cX, cY, cX, cY+20);
    ellipse(cX, cY, 10, 10);

    // Draw arms and legs based on pose
    if (walkPose == true) {
        line(cX-10, cY+10, cX+10, cY+10);
        line(cX, cY+20, cX-10, cY+30);
        line(cX, cY+20, cX+10, cY+30);
    } else {
        line(cX-10, cY+5, cX+10, cY+15);
        line(cX, cY+20, cX-5, cY+30);
        line(cX, cY+20, cX+5, cY+30);
    }
}

void keyPressed() {
    switch(keyCode) {
    case UP:
        walkPose = !walkPose;
        cY -= speed;
        break;
    case DOWN:
        walkPose = !walkPose;
        cY += speed;
        break;
    case LEFT:
        walkPose = !walkPose;
        cX -= speed;
        break;
    case RIGHT:
        walkPose = !walkPose;
        cX += speed;
        break;
    }
}
Equations of Motion (Simplified)

s = displacement
t = time
v = velocity
a = acceleration

• Constant acceleration (a)

\[ s_{i+1} = s_i + v_i \Delta t \]
\[ v_{i+1} = v_i + a \Delta t \]
float sx = 0.0;    // x position
float sy = 0.0;    // y position
float vx = 1.0;    // x velocity
float vy = 1.0;    // y velocity
float ay = 0.2;    // y acceleration (gravity)

void setup() {
    size(500, 500);
    fill(255, 0, 0);
    smooth();
    ellipseMode(CENTER);
}

void draw() {
    // Equations of Motion
    sx = sx + vx;
    sy = sy + vy;
    vy = vy + ay;

    // Bounce off walls
    if (sx <= 0.0 || sx >= width) vx = -vx;

    // Bounce off floor and
    // lose some velocity due to friction
    if (sy >= (height-10.0)) vy = -0.9*vy;

    // Draw at current location
    background(255);
    ellipse(sx, sy, 20, 20);
}
Iteration

Repetition of a program block
• Iterate when a block of code is to be repeated multiple times.

Options
• The while-loop
• The for-loop
Iteration: while-loop

while ( boolean_expression ) {
    statements;
    // continue;
    // break;
}

• Statements are repeatedly executed while the boolean expression continues to evaluate to true;
• To break out of a while loop, call break;
• To stop execution of statements and start again, call continue;
• All iterations can be written as while-loops.
void setup() {
    size(500, 500);
    smooth();

    float diameter = 500.0;
    while (diameter > 1.0) {
        ellipse(250, 250, diameter, diameter);
        diameter = diameter * 0.9;
    }
}

void draw() { }

What does this do?

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

    float diameter = 500.0;
    while (true) {
        ellipse(250, 250, diameter, diameter);
        diameter = diameter * 0.9;
        if (diameter <= 1.0) break;
    }
}

void draw() { }

while1.pde

while2.pde
Iteration: for-loop

for ( initialization; continuation_test; increment )
{
    statements;
    // continue;
    // break;
}

• A kind of iteration construct
• initialization, continuation test and increment commands are part of statement
• To break out of a while loop, call break;
• To stop execution of statements in block and start again, call continue;
void mousePressed() {

    for (int i = 0; i < 10; i++ )
    {
        print( i );
    }
    println();
}

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

  float diameter = 500.0;
  while ( diameter > 1.0 ) {
    ellipse( 250, 250, diameter, diameter);
    diameter = diameter - 10.0;
  }
}

void draw() { }

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

  for (float diameter = 500.0; diameter > 1.0; diameter -= 10.0 ) {
    ellipse( 250, 250, diameter, diameter);
  }
}

void draw() { }
Assignment #2 - Hints

• Decide what to draw based on the relative position of mouse and horizon line.
  – If mouse is above horizon, draw sky-appropriate things
  – If mouse is below horizon, draw ground-appropriate things

• Calculate a scale factor based on the distance of the mouse to horizon and if above or below.
  – Use built-in map() function to convert mouse y-position to a scale factor
  – Use scale factor to size the object being drawn
**map**

- A built-in function that maps some value from one range to another

```c
map(value, low1, high1, low2, high2);
```

- `map(100, 0, 500, 0, 1000);` → 200.0
- `map(250, 0, 500, -250, 250);` → 0.0
map
• A built-in function that maps some value from one range to another

map(value, low1, high1, low2, high2);

map(400, 200, 500, 0, 1); ➞ 0.66666667
map(150, 0, 200, 1, 0); ➞ 0.25
Pseudocode

• When the user clicks the mouse...
  – If the mouse's y-position is above the horizon
    • Use **one map function** to compute a scale factor that converts a range from the horizon to the top of the sketch (0.0) to a value between 0.0 and 1.0
    • Set the object type to a sky-appropriate thing
  – If the mouse's y-position is below the horizon
    • Use **a second map function** to compute a different scale factor that converts a range from the bottom of the sketch (height) to the horizon to a value between 1.0 and 0.0
    • Set the object type to a ground-appropriate thing
  – Use the mouse position and scale factor to draw appropriate object(s)
float delta = 5.0;
float factor = 0.0;

void setup() {

    size(500, 500);
}

void draw() {

    factor+=0.2;
    noStroke();

    for (float r=0.0; r<height; r+=delta) {
        for (float c=0.0; c<width; c+=delta) {

            // Use factor to scale shape
            float x = map(c, 0.0, 500.0, 0.0, 3.0*TWO_PI);
            float y = map(r, 0.0, 500.0, 0.0, 3.0*TWO_PI);
            float shade = map(sin(factor)*sin(x)*sin(y), -1.0, 1.0, 0, 255);

            // Use factor to shift shade
            //    float x = map(c, 0.0, 500.0, factor, factor+3.0*TWO_PI);
            //    float y = map(r, 0.0, 500.0, factor, factor+3.0*TWO_PI);
            //    float shade = map(sin(x)*sin(y), -1.0, 1.0, 0, 255);

            fill( shade );
            rect(r, c, delta, delta);
        }
    }
}

What does this do?