Processing Boot Camp
Control Structures
Creative Coding & Generative Art in Processing 2
Ira Greenberg, Dianna Xu, Deepak Kumar

Variables & Scope

Processing:

Variables & Scope

color color1 = color(227, 220, 0);
color color2 = color(37, 220, 0);
void setup() {
  // create and set up canvas
  size(300, 300);
  smooth();
  background(color1);
}
void draw() {
  fill(color2);
  square(mouseX, mouseY, 20);
}
void square(float x, float y, float side) {
  rectMode(CORNER);
  rect(x, y, side, side);
}

Variables & Scope

Global Variables
Either pre-defined
Or defined at top
Are visible everywhere
in this program

Local Variables
Either parameters
Or defined inside blocks
Are visible ONLY
in the block
After they are defined

Processing:
Math Functions

• Math functions return values:
Example:
void square(float x, float y, float side) {
  rectangle(0, 0, side, side);
}

• Processing has several pre-defined Math functions
for calculation, trigonometry, and random number
generation

Processing:
Pre-defined Math Functions

• Calculation
  abs(), ceil(), constrain(), dist(), exp(), floor(), lerp()
  log(), mag(), map(), max(), min(), norm(), pow()
  round(), sq(), sqrt()

• Trigonometry
  acos(), asin(), atan(), atan2(), cos(), degrees(),
  radians(), sin(), tan()

• Random
  noise(), noiseDetail(), noiseSeed(), random(),
  randomGaussian(), randomSeed()

Math Functions: Examples

• Calculation
  float a, b;
  a = exp(b);

• Trigonometry
  float rad = radians(45);
  float deg = degrees(PI/2);

• Random
  float x = random(100);  // return a random number [0.0..10.0)
  float y = random(6);   // return a random number [1.0, 6.0)
  int ix = int(random(10)); // return a random number [0..10)
  int iy = int(random(6)); // return a random number [1..6]
Example: Using random()

```java
void setup() {// Create and set canvas
  size(300, 300);
  smooth();
  background(255);
} // setup()

void draw() {
  stroke(0);
  rect(random(width),
       random(height),
       random(5, 20),
       random(5, 20));
} // draw();
```

2D Transformations: Translate

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

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

2D Transformations: Translate

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

Preserving Context

- `translate()` will change the coordinate system for the entire duration of the draw() cycle. It resets at each cycle.
- Use `pushMatrix()` and `popMatrix()` to preserve context during a draw() cycle. i.e.

```java
pushMatrix();
translate(<x>, <y>);
// Do something in the new coordinate context
popMatrix();
```

Example: House() again!

```java
void setup() {// Create and set canvas
  size(300, 300);
  smooth();
  background(187, 193, 127);
} // setup()

void draw() {
  // Draw a house at 50, 250 in 200x200 pixels
  house(50, 250, 200, 200);
} // draw()

void house(int houseX, int houseY, int houseWidth, int houseHeight) {
  // Draw a house at <houseX, houseY> (bottom left corner)
  // with width houseWidth and height houseHeight
  int wallHeight = houseHeight/2;  // height of wall is 1/2 of house height
  int roofHeight = houseHeight/2;
  int doorHeight = houseHeight/4;
  int doorWidth = houseWidth/4;

  pushMatrix();
  translate(houseX, houseY);
  // wall
  fill(206, 224, 14);
  rect(0, -wallHeight, houseWidth, wallHeight);
  // Draw Door
  fill(72, 26, 2);
  rect(houseWidth/2 - doorWidth/2, -doorHeight, doorWidth, doorHeight);
  // Draw roof
  fill(224, 14, 14);
  triangle(0, -wallHeight, houseWidth/2, -houseHeight, houseWidth, -wallHeight);
  popMatrix();
} // house()
```
Key Computing Ideas

• The computer follows a program’s instructions. There are four modes:
  - **Sequencing**
    All statements are executed in sequence
  - **Function Application**
    Control transfers to the function when invoked
    Control returns to the statement following upon return
  - **Repetition**
    Enables repetitive execution of statement blocks
  - **Selection**
    Enables choice among a block of statements

• All computer algorithms/programs utilize these modes.

Sequencing

• Refers to sequential execution of a program’s statements

```plaintext
do this;  size(200,200);
then do this; background(1255);
and then do this; stroke(128);
etc.  rect(20, 20, 40, 40);
```

Function Application

• Control transfers to the function when invoked
• Control returns to the statement following upon return

```plaintext
void draw() {
    // Draw a house at 50, 250 in 200x200 pixels
    house(50, 250, 200, 200);
    house(20, 100, 50, 50);
    house(230, 100, 50, 75);
} // draw()

void house(int houseX, int houseY, int houseWidth, int houseHeight) {
    // Draw a house at (houseX, houseY) (bottom left corner)
    // with width houseWidth and height houseHeight
    // with some additional details...
} // house()
```

Repetition

• Enables repetitive execution of statement blocks

```plaintext
lather
rinse
repeat
```

Loops: Controlled Repetition

• **While Loop**
  ```plaintext
  while (condition) {
      stuff to repeat
  }
  ```

• **Do-While Loop**
  ```plaintext
  do stuff to repeat
  while (condition)
  ```

• **For Loop**
  ```plaintext
  for (initial; condition; update) {
      stuff to repeat
  }
  ```
Loops: Controlled Repetition

- **While Loop**
  
  ```
  while (condition) {
      stuff to repeat
  }
  ```

- **Do-While Loop**
  
  ```
  do {
      stuff to repeat
  } while (condition);
  ```

- **For Loop**
  
  ```
  for (<init>; <condition>; <update>) {
      stuff to repeat
  }
  ```

  All of these repeat the stuff in the block. The block `{…}` is called the Loop’s Body.

Conditions

- Conditions are **boolean** expressions.
- Their value is either **true** or **false**
  
  e.g.
  
  POTUS is a woman: **false**
  
  5 is greater than 3: **true**
  
  5 is less than 3: **false**

Writing Conditions in Processing

- Boolean expressions can be written using **boolean operators**.

  Here are some simple expressions...

<table>
<thead>
<tr>
<th>Expression</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;</code></td>
<td>less than</td>
</tr>
<tr>
<td><code>&lt;=</code></td>
<td>less than/equal to</td>
</tr>
<tr>
<td><code>==</code></td>
<td>equal to</td>
</tr>
<tr>
<td><code>!=</code></td>
<td>not equal to</td>
</tr>
<tr>
<td><code>&gt;</code></td>
<td>greater than</td>
</tr>
<tr>
<td><code>&gt;=</code></td>
<td>greater than/equal to</td>
</tr>
</tbody>
</table>

  Here are some examples of simple expressions:

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>X &lt; Y</th>
<th>X &lt;= Y</th>
<th>X == Y</th>
<th>X != Y</th>
<th>X &gt; Y</th>
<th>X &gt;= Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>

Logical Operations

- Combine two or more simple boolean expressions using logical operators:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&amp;&amp;</code></td>
<td>and (x &lt; y) &amp; (y &lt; z)</td>
</tr>
<tr>
<td>`</td>
<td></td>
</tr>
<tr>
<td><code>!</code></td>
<td>not (x &lt; y)</td>
</tr>
</tbody>
</table>

  Here are the truth tables:

| A | B | A && B | A || B | !A |
|---|---|-------|-------|----|
| F | F |  F    |  F    | T  |
| F | T |  T    |  T    | T  |
| T | F |  F    |  T    | F  |
| T | T |  T    |  T    | F  |
Conditions in While Loops

```cpp
while (<condition>) {
  stuff to repeat
}
```

```
ask x = 0;
while (x < width) {
  ellipse(x, height/2, 50, 50);
  x = x + 55;
}
```

Loops: Controlled Repetition

- **While Loop**
  ```cpp
  while (<condition>) {
    stuff to repeat
  }
  ```

- **Do-While Loop**
  ```cpp
  do {
    stuff to repeat
  } while (<condition>)
  ```

- **For Loop**
  ```cpp
  for (<init>;<condition>;<update>) {
    stuff to repeat
  }
  ```

Do-While Loops

```cpp
do {
  stuff to repeat
} while (<condition>);
```

Loops: Critical Components

- **Loop initialization**
  Things to do to set up the repetition

- **Loop Termination Condition**
  When to terminate the loop

- **Loop Body**
  The stuff to be repeated

- **Loop update**
  For the next repetition/iteration

For Loops

```cpp
for (<init>;<condition>;<update>) {
  stuff to repeat
}
```
Loops: Critical Components

- **Loop initialization**
  Things to do to set up the repetition

- **Loop Termination Condition**
  When to terminate the loop

- **Loop Body**
  The stuff to be repeated

- **Loop update**
  For the next repetition/iteration

---

Key Computing Ideas

- The computer follows a program’s instructions. There are four modes:
  - **Sequencing**
    All statements are executed in sequence
  - **Function Application**
    Control transfers to the function when invoked
    Control returns to the statement following upon return
  - **Repetition**
    Enables repetitive execution of statement blocks
  - **Selection**
    Enables choice among a block of statements

- All computer algorithms/programs utilize these modes.
Selection

- Enables choice among a block of statements
  
  Should I...
  { study }
  { sleep }
  { watch a movie }
  { veg out }
  { etc. }

- **If-statements** are one way of doing this

---

Selection: If Statement

```c
if (<condition>) {
  do this
}
else {
  do that
}
else if (<condition>) {
  do that
}
else if (…) {
  …
}
else {
  whatever it is you wanna do
}
```

At most ONE block is selected and executed.

---

Examples with if...

---