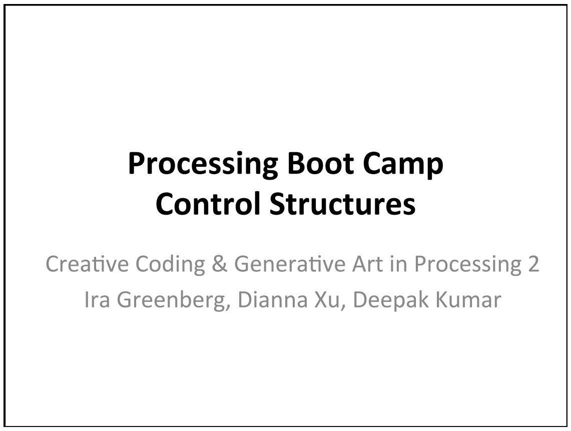
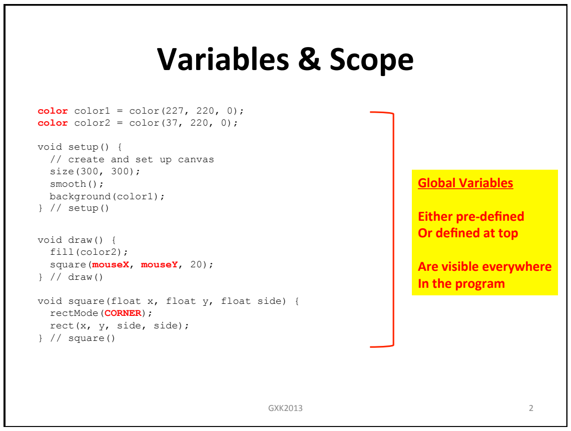


Processing Boot Camp
Control Structures

Creative Coding & Generative Art in Processing 2

Ira Greenberg, Dianna Xu, Deepak Kumar



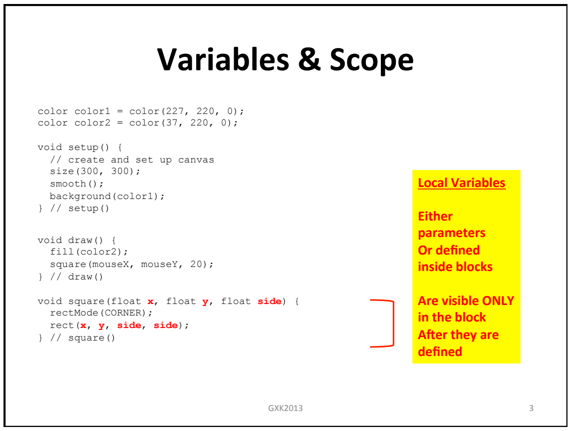
Variables & Scope

```
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);
} // setup()


void draw() {
  fill(color2);
  square(mouseX, mouseY, 20);
} // draw()

void square(float x, float y, float side) {
  rectMode(CORNER);
  rect(x, y, side, side);
} // square()
```

Global Variables

Either pre-defined
Or defined at top

Are visible everywhere
In the program



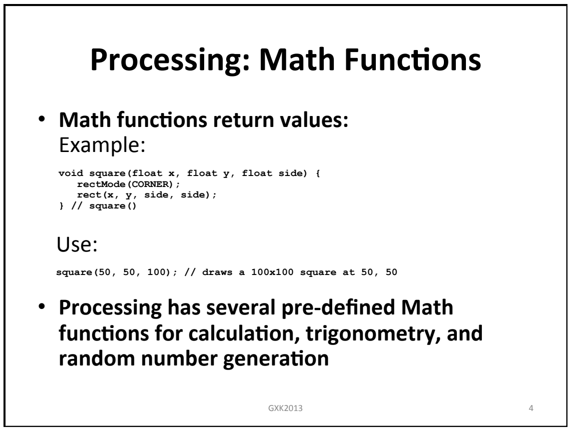
Variables & Scope

```
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);
} // setup()


void draw() {
  fill(color2);
  square(mouseX, mouseY, 20);
} // draw()

void square(float x, float y, float side) {
  rectMode(CORNER);
  rect(x, y, side, side);
} // square()
```

Local Variables

Either
parameters
Or defined
inside blocks

Are visible ONLY
in the block
After they are
defined



Processing: Math Functions

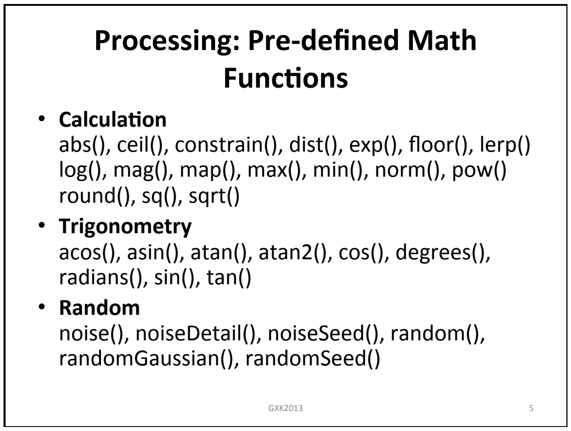
- **Math functions return values:**
  Example:

```
void square(float x, float y, float side) {
  rectMode(CORNER);
  rect(x, y, side, side);
} // square()
```

Use:

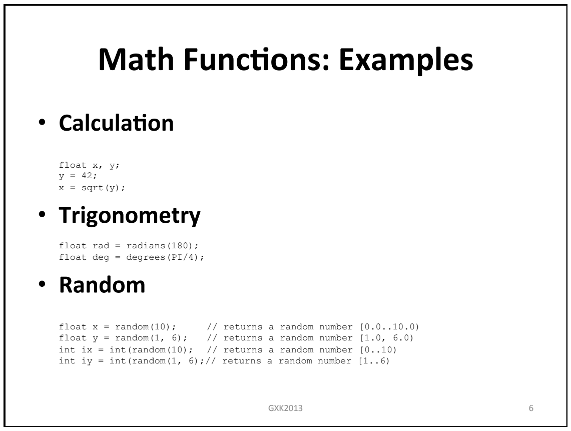
```
square(50, 50, 100); // draws a 100x100 square at 50, 50
```

- **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 x, y;
y = 42;
x = sqrt(y);
```

- **Trigonometry**

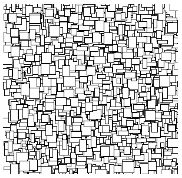
```
float rad = radians(180);
float deg = degrees(PI/4);
```

- **Random**

```
float x = random(10);     // returns a random number [0.0..10.0)
float y = random(1, 6);   // returns a random number [1.0, 6.0)
int ix = int(random(10);  // returns a random number [0..10)
int iy = int(random(1, 6);// returns a random number [1..6)
```

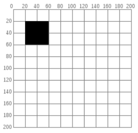
Example: Using random()

```
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();
```



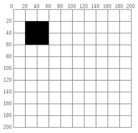
2D Transformations: Translate

rect(20, 20, 40, 40);

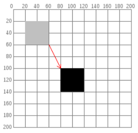


2D Transformations: Translate

rect(20, 20, 40, 40);



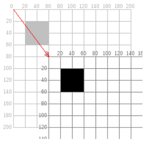
rect(20+60, 20+80, 40, 40);



2D Transformations: Translate

translate(60, 80);

rect(20, 20, 40, 40);



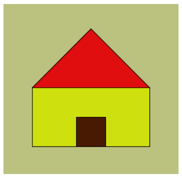
Preserving Context

- **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.

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

Example: House() again!

```
// Draw a simple house

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

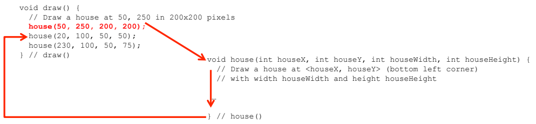
- Refers to sequential execution of a program's statements

```
do this;
then do this;
and then do this;
etc.
```

```
size(200,200);
background(255);

stroke(128);
rect(20, 20, 40, 40);
```

Function Application

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

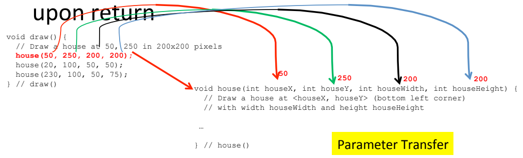


```
void draw() {
  // Draw a house at 50, 250 in 200x200 pixels
  house(50, 250, 200, 200);
  house(20, 100, 50, 50);
  house(230, 100, 50, 75);
} // draw()
```

```
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

…

} // house()
```

Function Application

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



```
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()
```

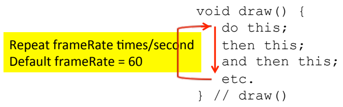
```
void house(int houseX, int houseY, int houseWidth, int houseHeight) {
  // Draw a house at <houseX, houseY> (bottom left corner)
  // with width houseWidth and height houseHeight

 …

} // house()
```

Parameter Transfer

Repetition

- Enables repetitive execution of statement blocks

```
lather
rinse
repeat
```



Repeat frameRate times/second
Default frameRate = 60

```
void draw() {
   do this;
   then this;
   and then this;
   etc.
} // draw()
```

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
}
```

Loops: Controlled Repetition

- **While Loop**

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

- **Do-While Loop**

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

- **For Loop**

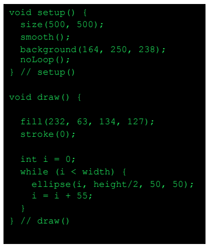
```
for (<init>; <condition>; <update>) {
   stuff to repeat
}
```

All of these repeat
the stuff in the block

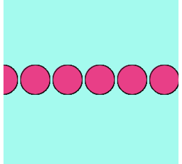
The block
{…}
is called the Loop's Body

While Loops

while (*<condition>*) {
stuff to repeat
}



```
void setup() {
  size(500, 500);
  smooth();
  background(164, 250, 238);
  noLoop();
} // setup()

void draw() {

  fill(232, 63, 134, 127);
  stroke(0);

  int i = 0;
  while (i < width) {
    ellipse(i, height/2, 50, 50);
    i = i + 55;
  }
} // draw()
```



Conditions

- Conditions are **boolean** expressions.
- Their value is either **true** or **false**
  e.g.

POTUS is a woman

5 is greater than 3

5 is less than 3

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…

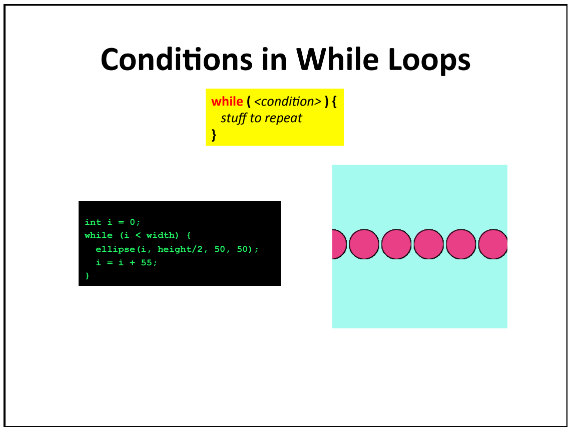
<	less than	5 < 3
<=	less than/equal to	x <= y
==	equal to	x == (y+j)
!=	not equal to	x != y
>	greater than	x > y
>=	greather than/equal to	x >= y

Logical Operations

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

&&	and	(x < y) && (y < z)
||	or	(x < y) || (x < z)
!	not	! (x < y)

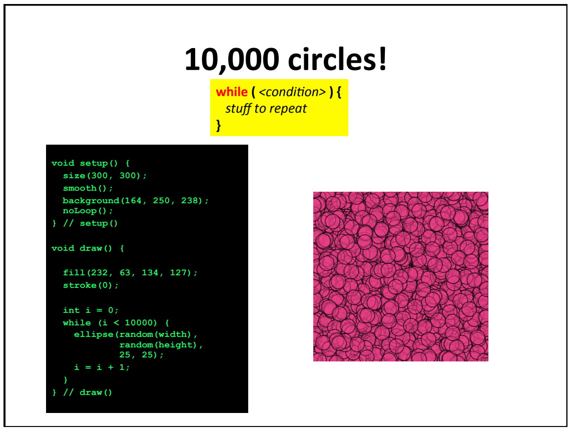
A	B	A && B	A || B	!A
false	false	false	false	true
false	true	false	true	true
true	false	false	true	false
true	true	true	true	false



Conditions in While Loops

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

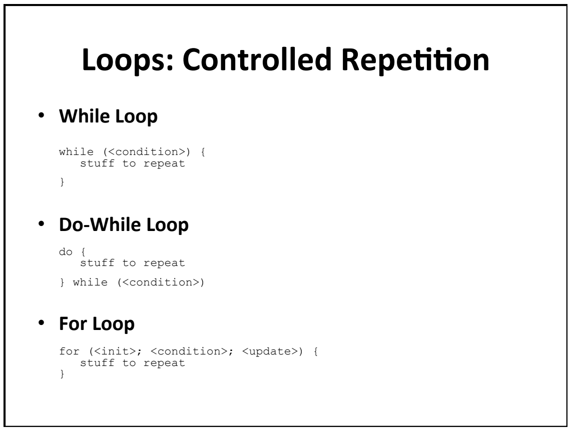
```
int i = 0;
while (i < width) {
  ellipse(i, height/2, 50, 50);
  i = i + 55;
}
```



10,000 circles!

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

```
void setup() {
  size(300, 300);
  smooth();
  background(164, 250, 238);
  noLoop();
} // setup()

void draw() {

  fill(232, 63, 134, 127);
  stroke(0);

  int i = 0;
  while (i < 10000) {
    ellipse(random(width),
            random(height),
            25, 25);
    i = i + 1;
  }
} // draw()
```



Loops: Controlled Repetition

- **While Loop**

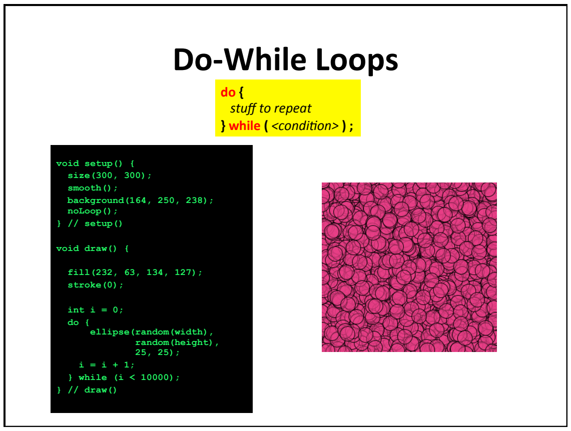
```
while (<condition>) {
   stuff to repeat
}
```

- **Do-While Loop**

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

- **For Loop**

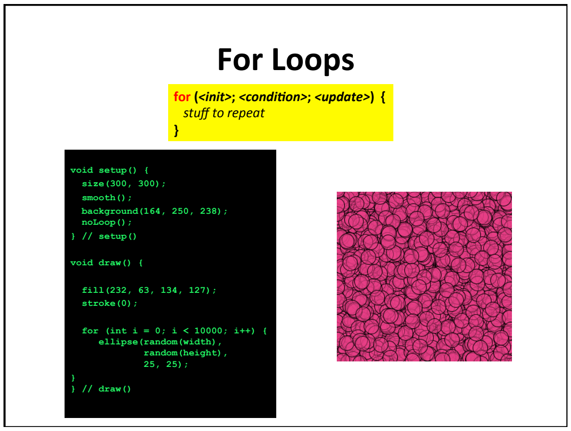
```
for (<init>; <condition>; <update>) {
   stuff to repeat
}
```



Do-While Loops

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

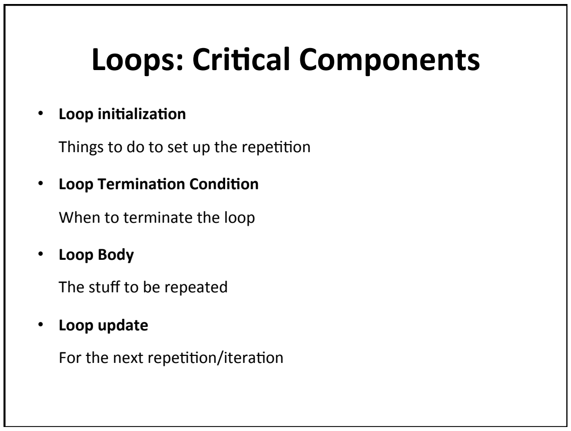
```
void setup() {
  size(300, 300);
  smooth();
  background(164, 250, 238);
  noLoop();
} // setup()

void draw() {

  fill(232, 63, 134, 127);
  stroke(0);

  int i = 0;
  do {
      ellipse(random(width),
              random(height),
              25, 25);
    i = i + 1;
  } while (i < 10000);
} // draw()
```



For Loops

```
for (<init>; <condition>; <update>) {
  stuff to repeat
}
```

```
void setup() {
  size(300, 300);
  smooth();
  background(164, 250, 238);
  noLoop();
} // setup()

void draw() {

  fill(232, 63, 134, 127);
  stroke(0);

  for (int i = 0; i < 10000; i++) {
     ellipse(random(width),
             random(height),
             25, 25);
}
} // draw()
```



Loops: Critical Components

- **Loop initialization**

  Things to do to set up the repetition

- **Loop Termination Condition**

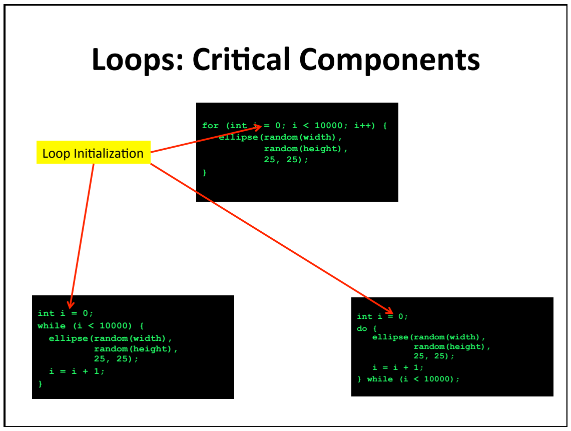
  When to terminate the loop

- **Loop Body**

  The stuff to be repeated

- **Loop update**

  For the next repetition/iteration

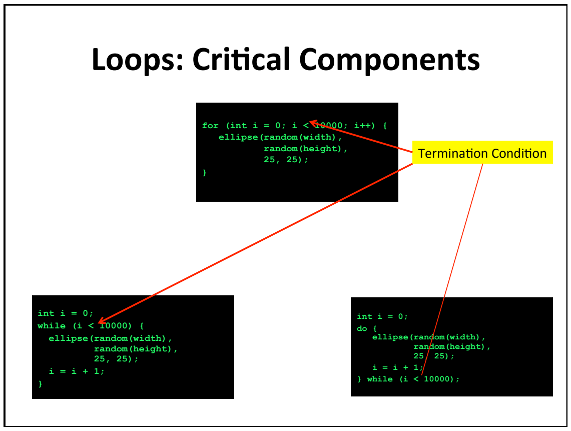


Loops: Critical Components

```
for (int i = 0; i < 10000; i++) {
   ellipse(random(width),
           random(height),
           25, 25);
}
```

```
int i = 0;
while (i < 10000) {
  ellipse(random(width),
          random(height),
          25, 25);
  i = i + 1;
}
```

```
int i = 0;
do {
   ellipse(random(width),
           random(height),
           25, 25);
   i = i + 1;
} while (i < 10000);
```

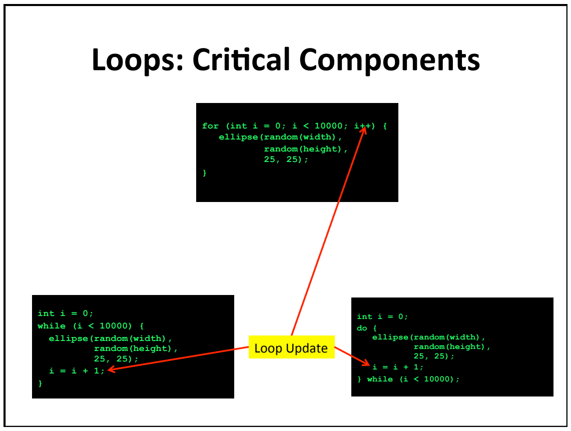


Loops: Critical Components

```
for (int i = 0; i < 10000; i++) {
   ellipse(random(width),
           random(height),
           25, 25);
}
```

```
int i = 0;
while (i < 10000) {
  ellipse(random(width),
          random(height),
          25, 25);
  i = i + 1;
}
```

```
int i = 0;
do {
   ellipse(random(width),
           random(height),
           25, 25);
   i = i + 1;
} while (i < 10000);
```

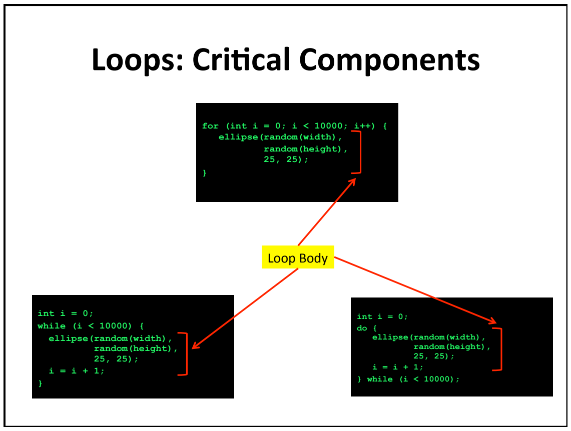


Loops: Critical Components

```
for (int i = 0; i < 10000; i++) {
   ellipse(random(width),
           random(height),
           25, 25);
}
```

```
int i = 0;
while (i < 10000) {
  ellipse(random(width),
          random(height),
          25, 25);
  i = i + 1;
}
```

```
int i = 0;
do {
   ellipse(random(width),
           random(height),
           25, 25);
   i = i + 1;
} while (i < 10000);
```



Loops: Critical Components

```
for (int i = 0; i < 10000; i++) {
   ellipse(random(width),
           random(height),
           25, 25);
}
```

```
int i = 0;
while (i < 10000) {
  ellipse(random(width),
          random(height),
          25, 25);
  i = i + 1;
}
```

```
int i = 0;
do {
   ellipse(random(width),
           random(height),
           25, 25);
   i = i + 1;
} while (i < 10000);
```

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

What happens when any one of these is missing or incorrectly encoded??

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

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

```
if ( <condition> ) {
  do this
}
else {
  do that
}
```

```
if ( <condition> ) {
  do this
}
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…