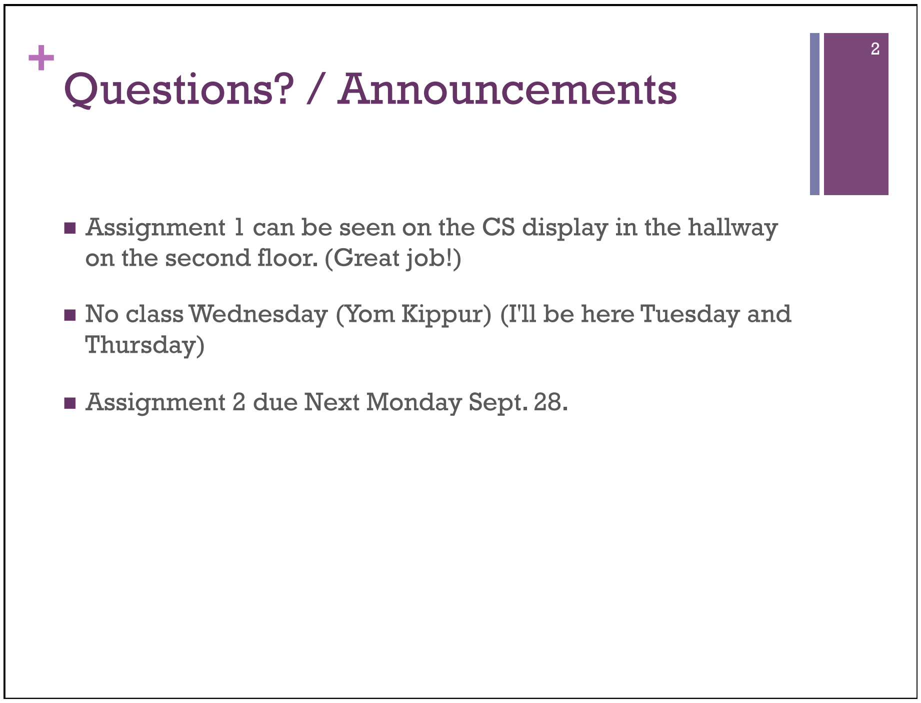
+

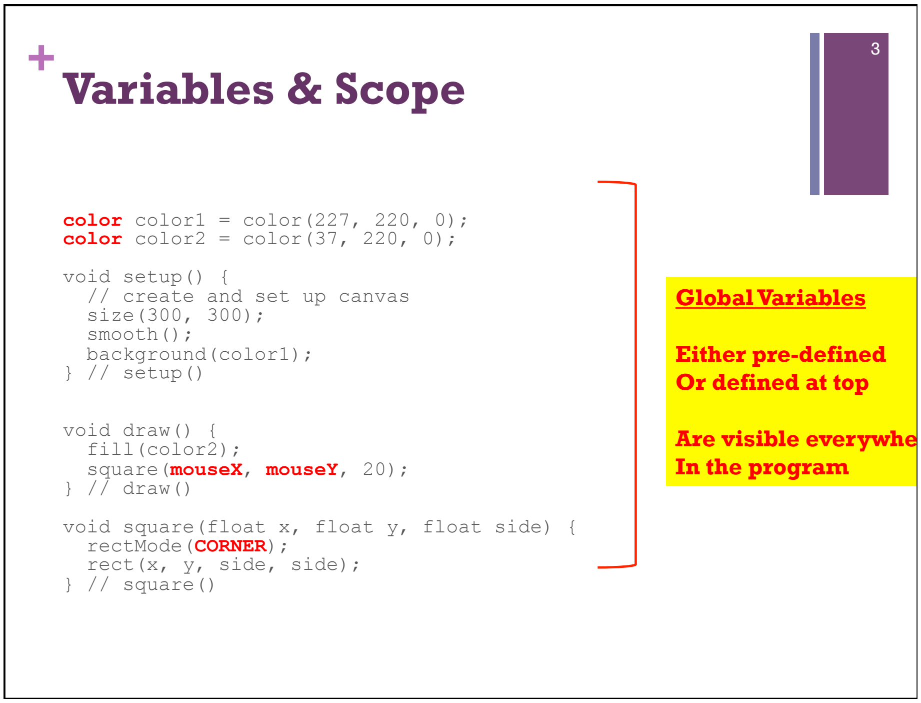
Decisions and Control Structure



+

Questions? / Announcements

- Assignment 1 can be seen on the CS display in the hallway on the second floor. (Great job!)
- No class Wednesday (Yom Kippur) (I'll be here Tuesday and Thursday)
- Assignment 2 due Next Monday Sept. 28.



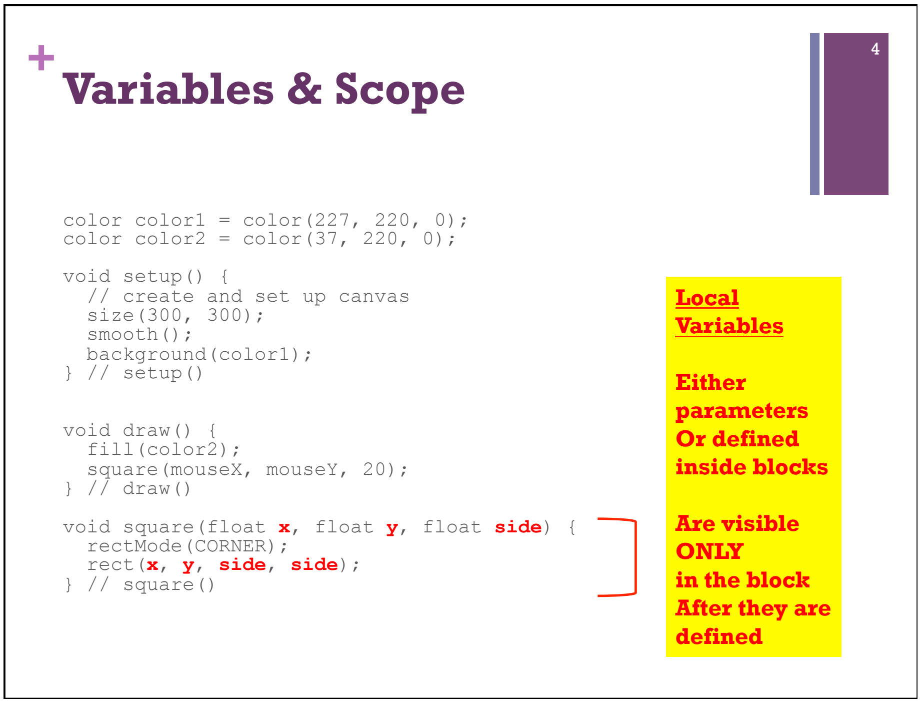
+ Variables & Scope

```
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 everywhe
In the program



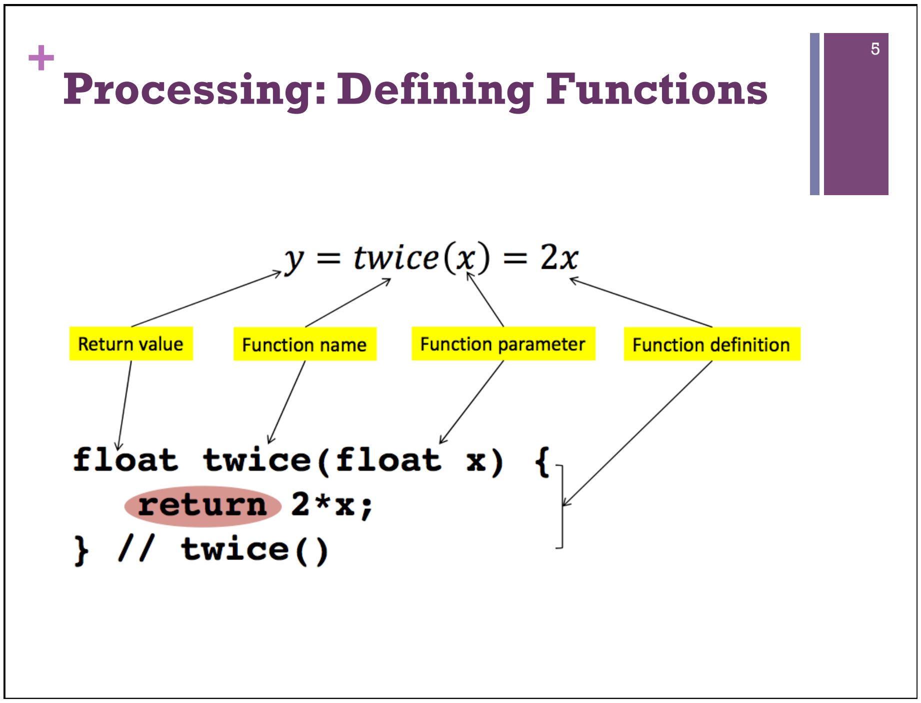
+ Variables & Scope

```
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

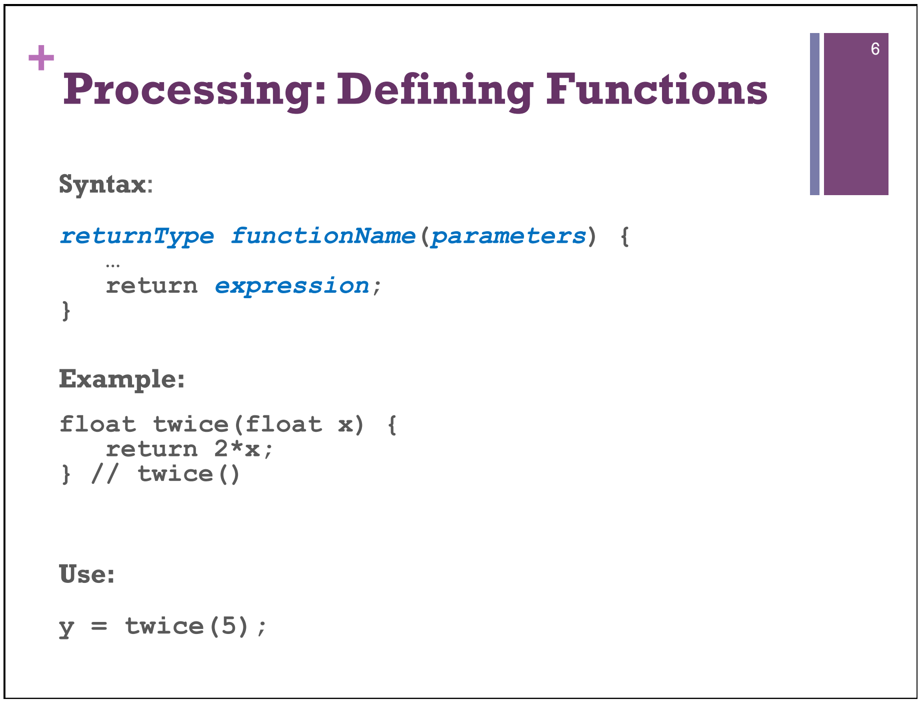
Are visible
ONLY
in the block
After they are
defined



Processing: Defining Functions

$$y = twice(x) = 2x$$

```
float twice(float x) {
   return 2*x;
} // twice()
```



Processing: Defining Functions

Syntax:

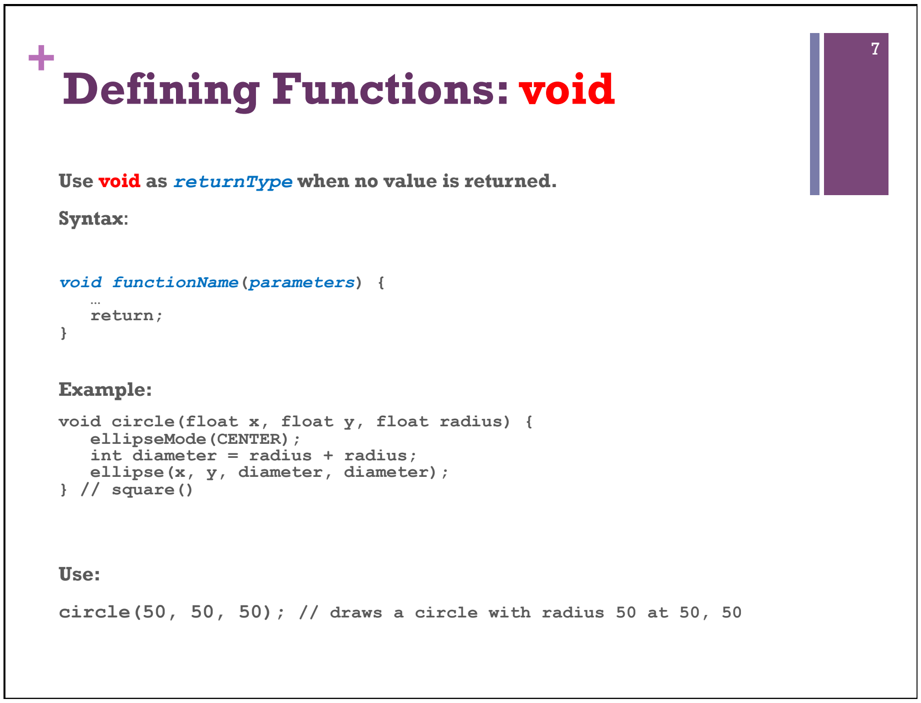
```
returnType functionName(parameters) {
   …
   return expression;
}
```

Example:

```
float twice(float x) {
   return 2*x;
} // twice()
```

Use:

```
y = twice(5);
```



+ 7

Defining Functions: void

Use void as *returnType* when no value is returned.

Syntax:

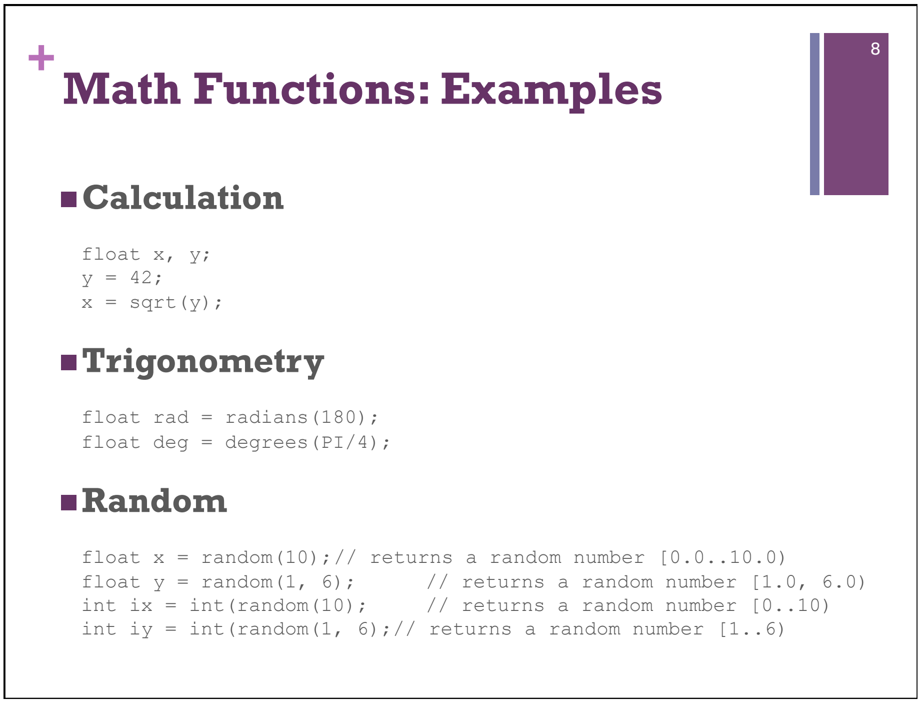
```
void functionName(parameters) {
   …
   return;
}
```

Example:

```
void circle(float x, float y, float radius) {
   ellipseMode(CENTER);
   int diameter = radius + radius;
   ellipse(x, y, diameter, diameter);
} // square()
```

Use:

```
circle(50, 50, 50); // draws a circle with radius 50 at 50, 50
```



+ 8

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)
```

9

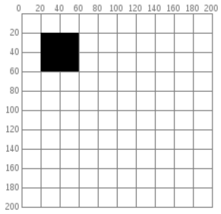
+ Example: Using random()

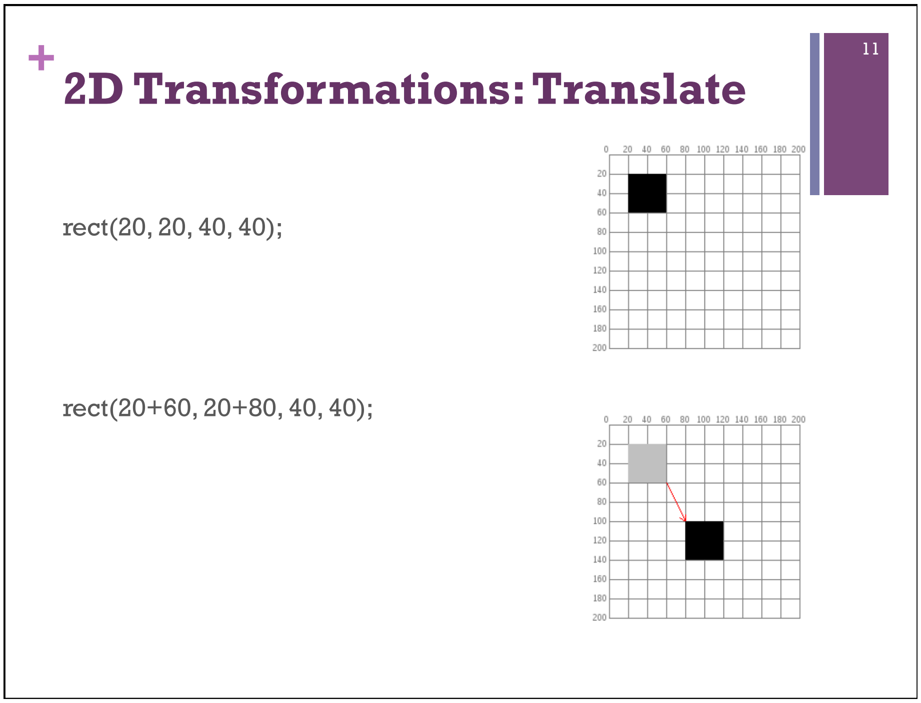
```
void setup() {// Create and set canvas
  size(300, 300);
  smooth();
  background(255);
} // setup()

void draw() {
  stroke(0);
  fill(random(255),
       random(255),
       random(255));
  ellipse(random(width),
  random(height),
  random(5, 20),
  random(5, 20));
} // draw();
```

10

+ 2D Transformations: Translate

rect(20, 20, 40, 40);



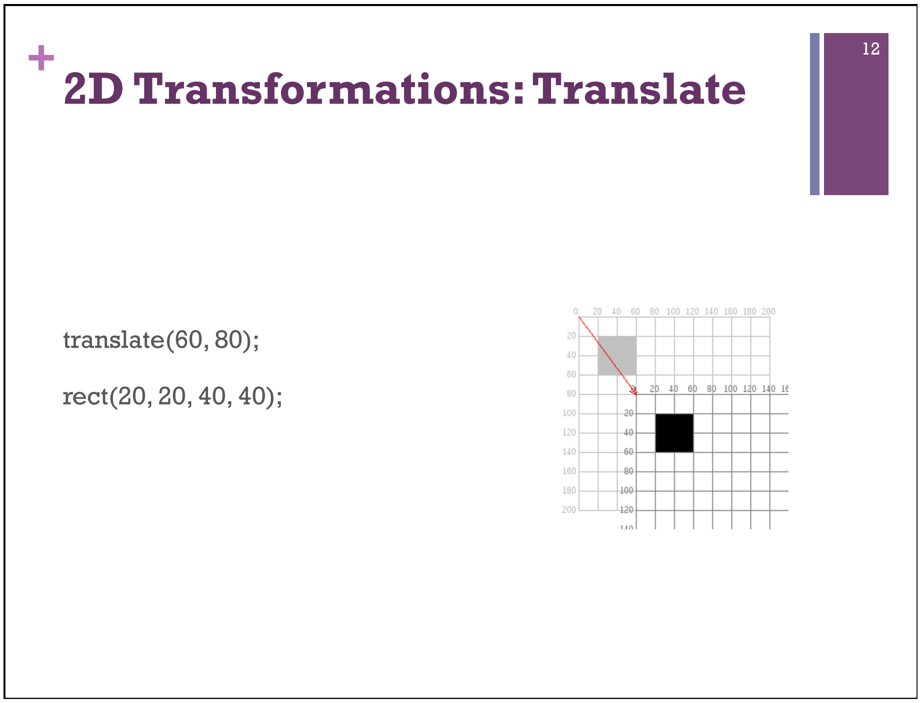


11

+ 2D Transformations: Translate

rect(20, 20, 40, 40);

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



12

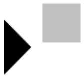
+ 2D Transformations: Translate

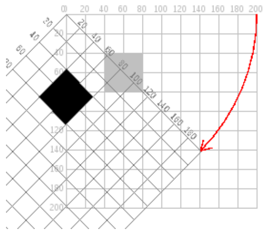
translate(60, 80);

rect(20, 20, 40, 40);

+ 2D Transformations: Rotate

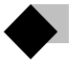
```
void setup() {
  size(200, 200);
  background(255);
  smooth();
  fill(192);
  noStroke();

  rect(40, 40, 40, 40);

  pushMatrix();
  rotate(radians(45));
  fill(0);
  rect(40, 40, 40, 40);
  popMatrix();
} // setup()
```

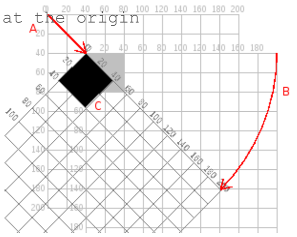


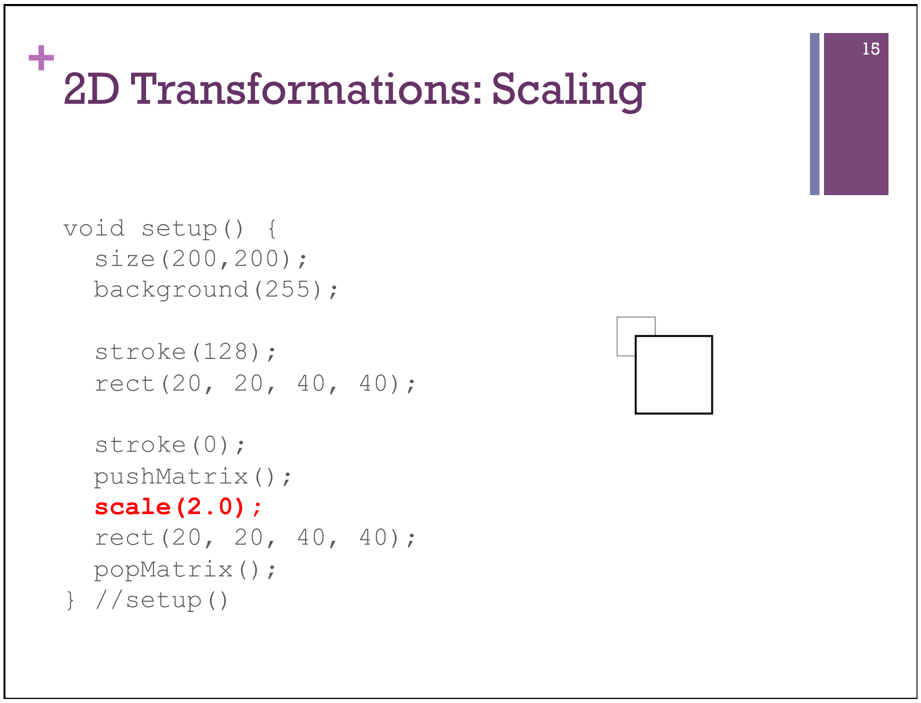


+ 2D Transformations: Rotate

```
void setup() {
  size(200, 200);
  background(255);
  smooth();
  fill(192);
  noStroke();

  rect(40, 40, 40, 40);

  pushMatrix(); // move the origin to the pivot point
  translate(40, 40); // then pivot the grid
  rotate(radians(45)); // and draw the square at the origin
  fill(0);
  rect(0, 0, 40, 40);
  popMatrix();
} // setup()
```

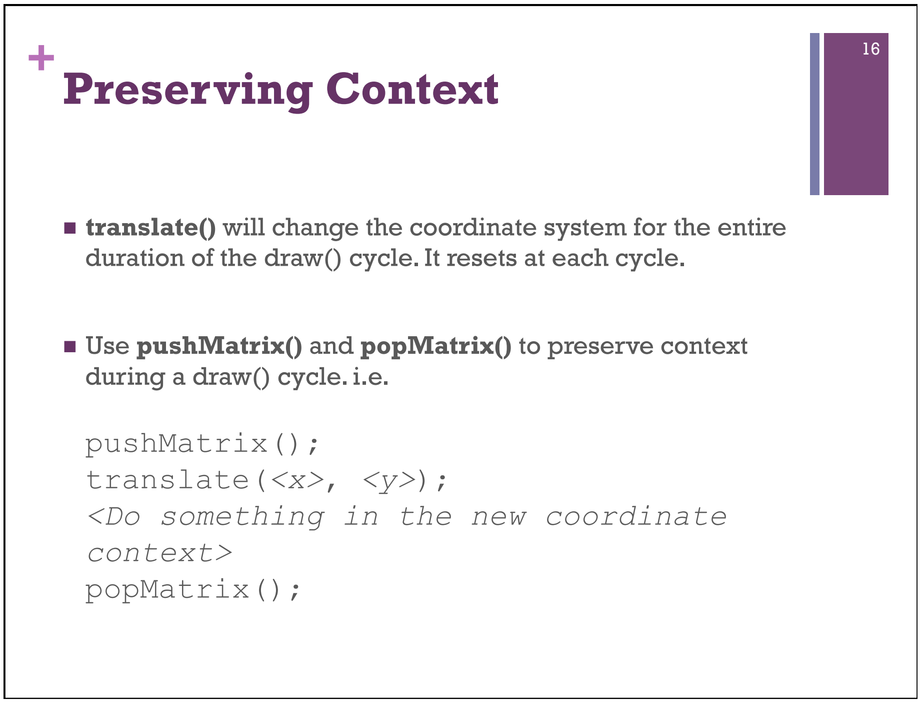






+ 2D Transformations: Scaling

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

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

  stroke(0);
  pushMatrix();
  scale(2.0);
  rect(20, 20, 40, 40);
  popMatrix();
} //setup()
```

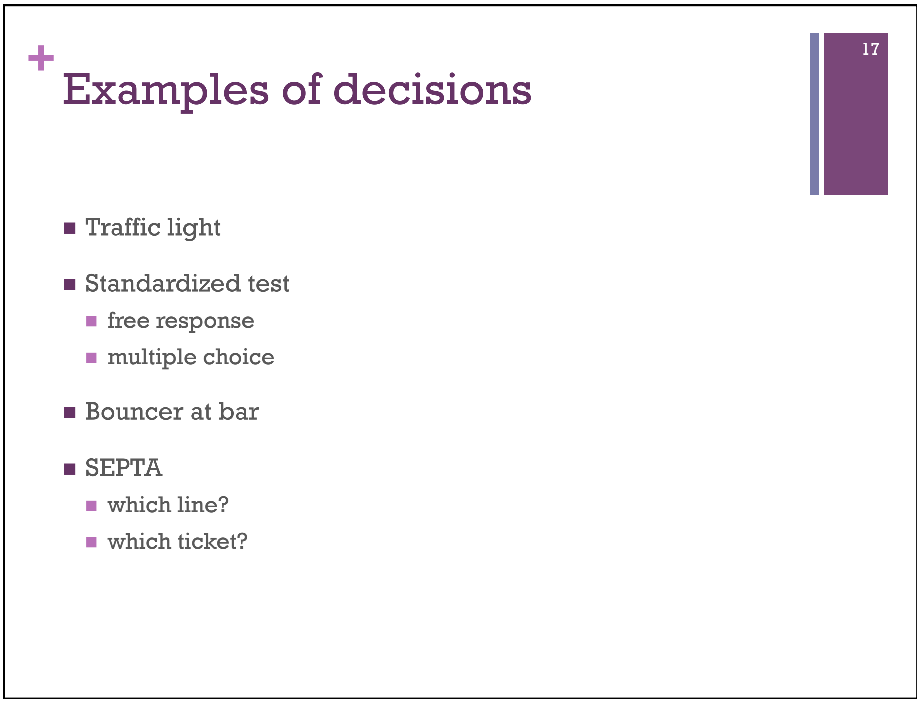


+ Preserving Context

- **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.

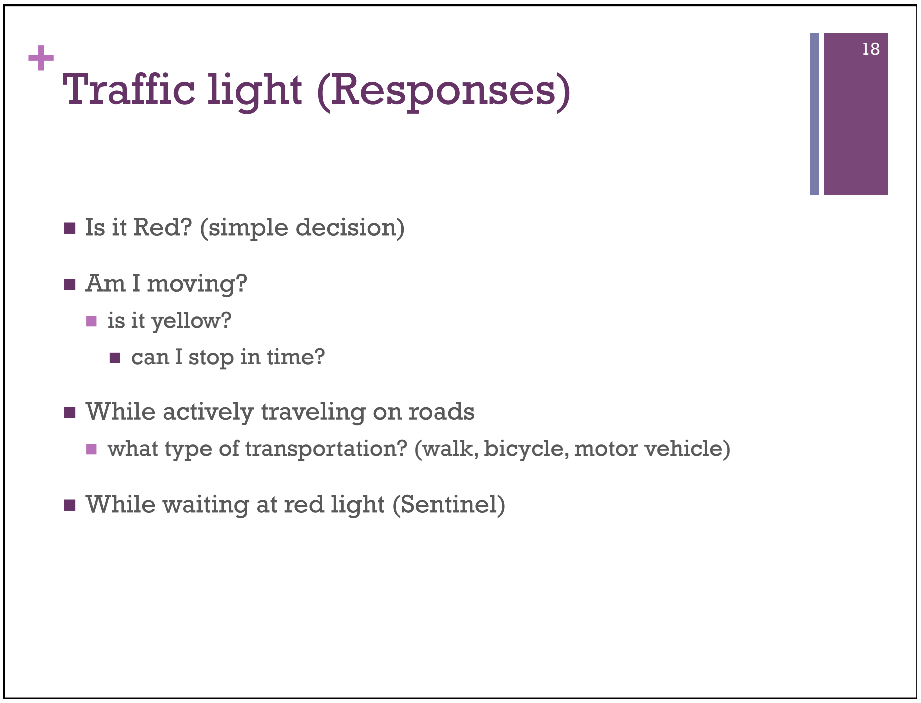
```
pushMatrix();
translate(<x>, <y>);
<Do something in the new coordinate
context>
popMatrix();
```



+ Examples of decisions

17

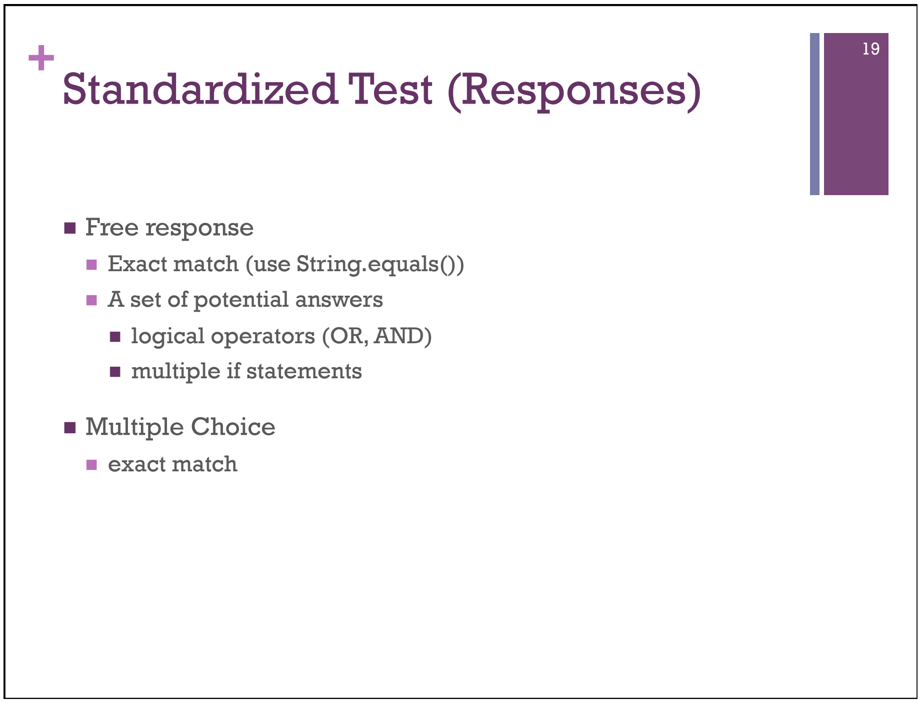
- Traffic light
- Standardized test
  - free response
  - multiple choice
- Bouncer at bar
- SEPTA
  - which line?
  - which ticket?



+ Traffic light (Responses)

18

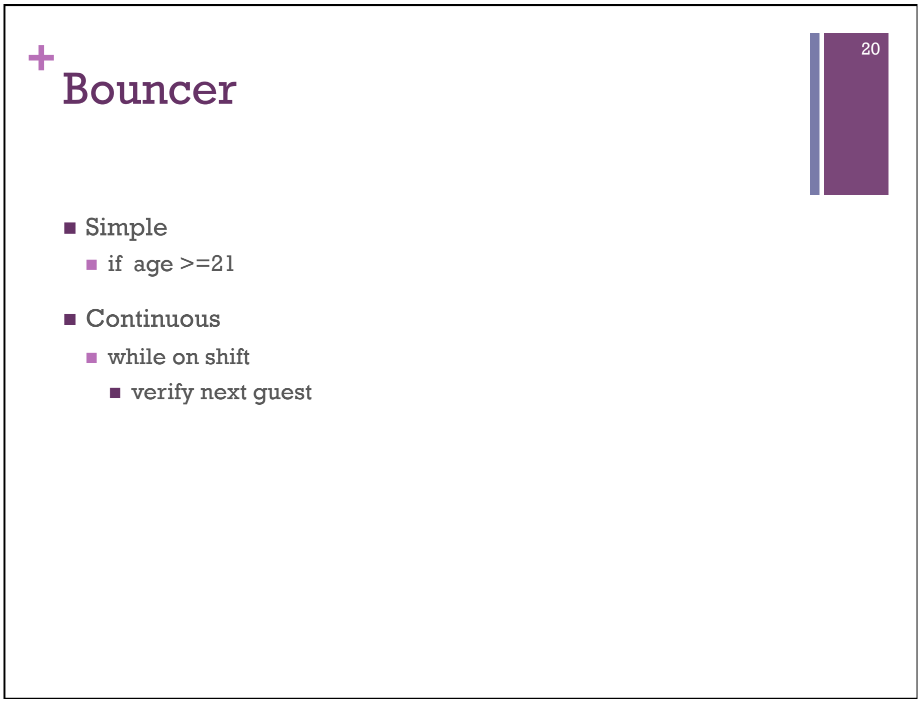
- Is it Red? (simple decision)
- Am I moving?
  - is it yellow?
    - can I stop in time?
- While actively traveling on roads
  - what type of transportation? (walk, bicycle, motor vehicle)
- While waiting at red light (Sentinel)



19

Standardized Test (Responses)

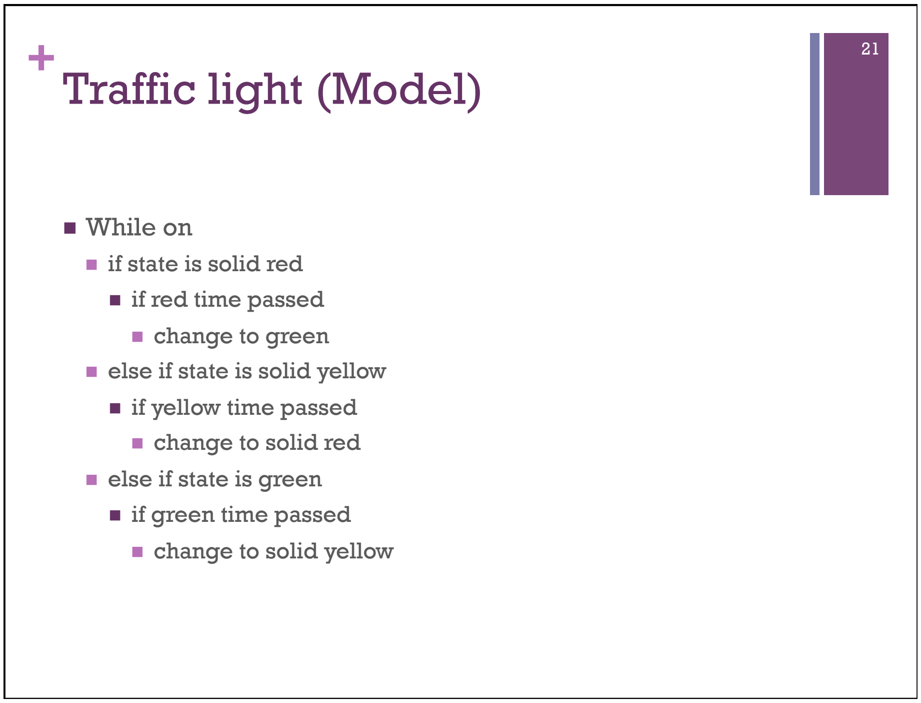
- Free response
  - Exact match (use String.equals())
  - A set of potential answers
    - logical operators (OR, AND)
    - multiple if statements
- Multiple Choice
  - exact match



20

Bouncer

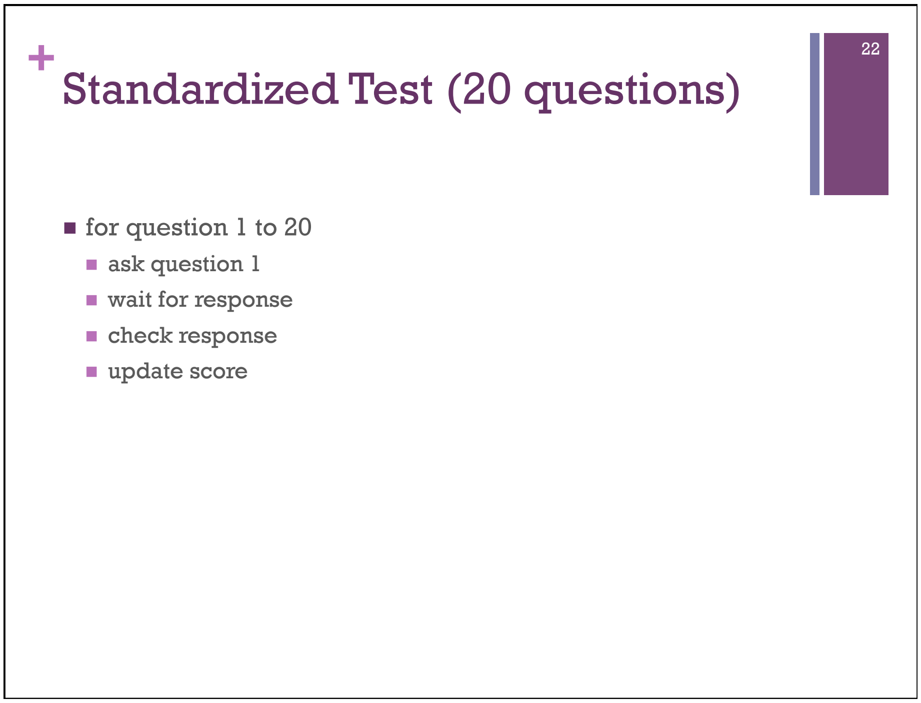
- Simple
  - if age >=21
- Continuous
  - while on shift
    - verify next guest



21

+ Traffic light (Model)

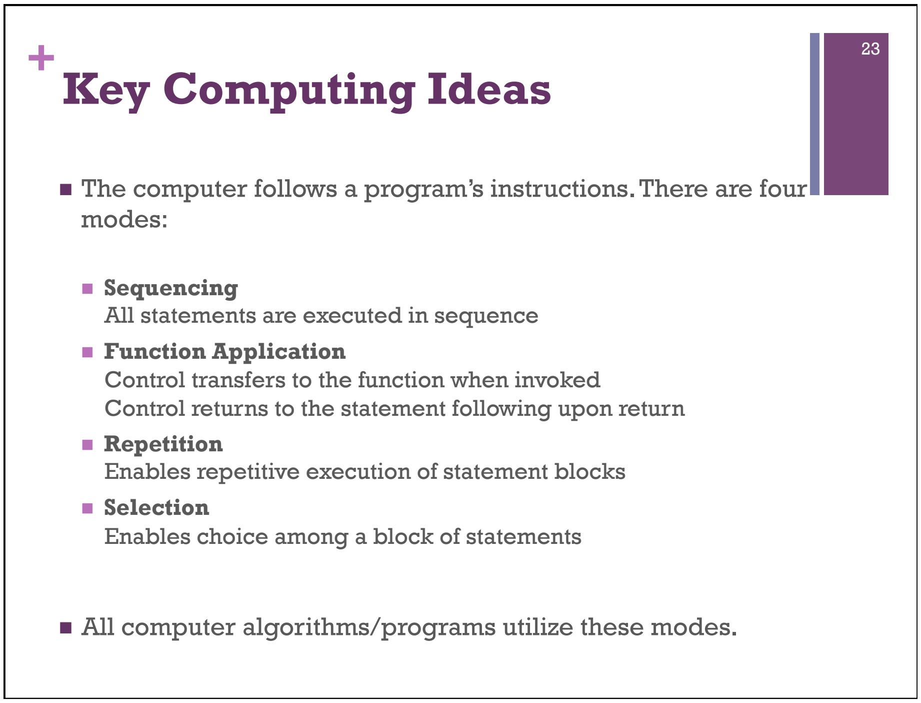
- While on
  - if state is solid red
    - if red time passed
      - change to green
  - else if state is solid yellow
    - if yellow time passed
      - change to solid red
  - else if state is green
    - if green time passed
      - change to solid yellow



22

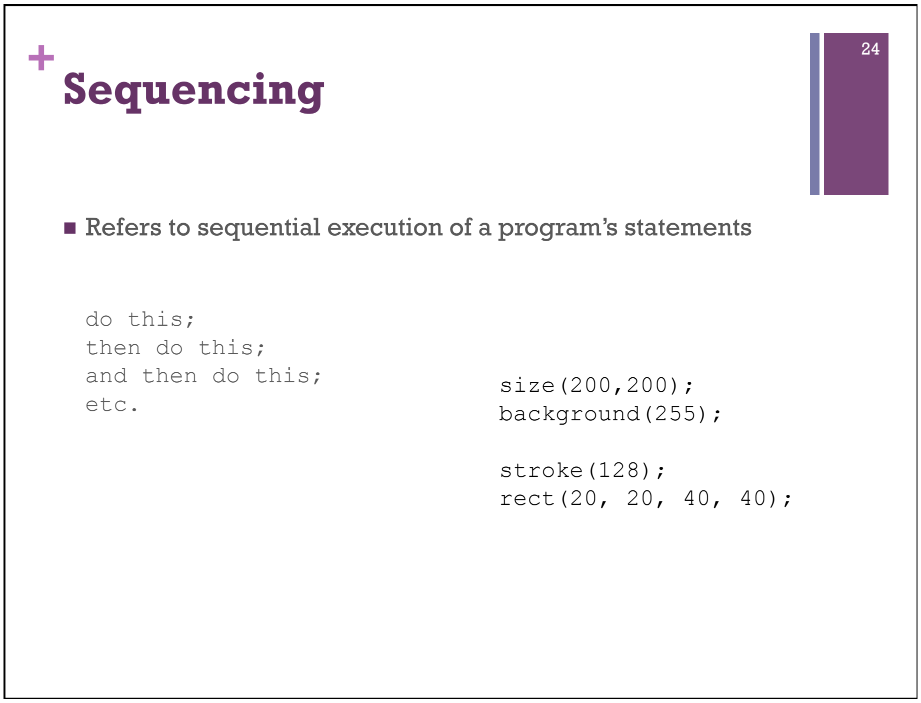
+ Standardized Test (20 questions)

- for question 1 to 20
  - ask question 1
  - wait for response
  - check response
  - update score



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.

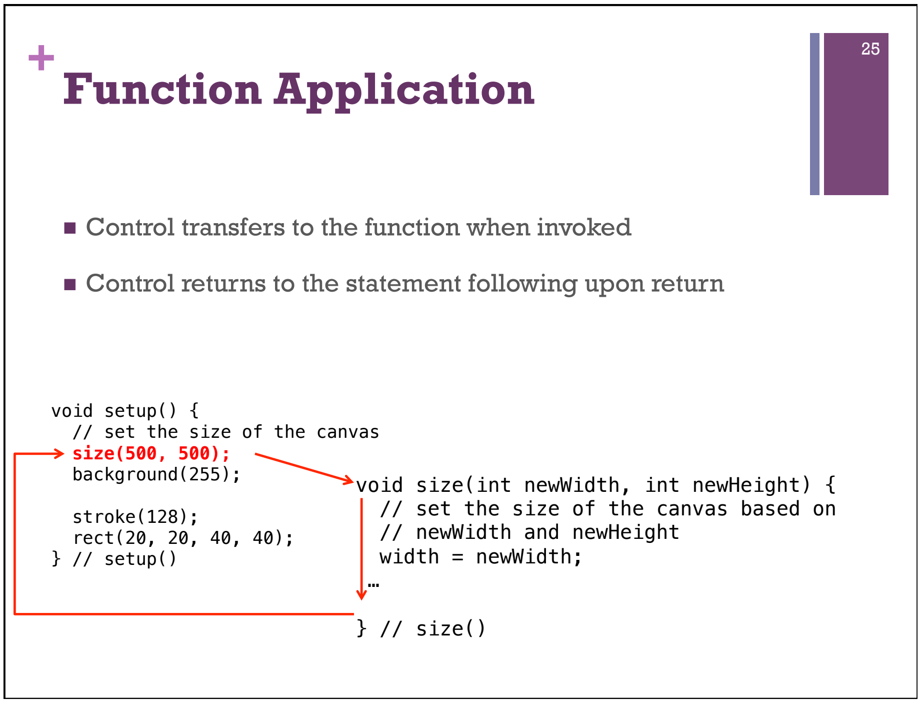


Sequencing

- 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);
```



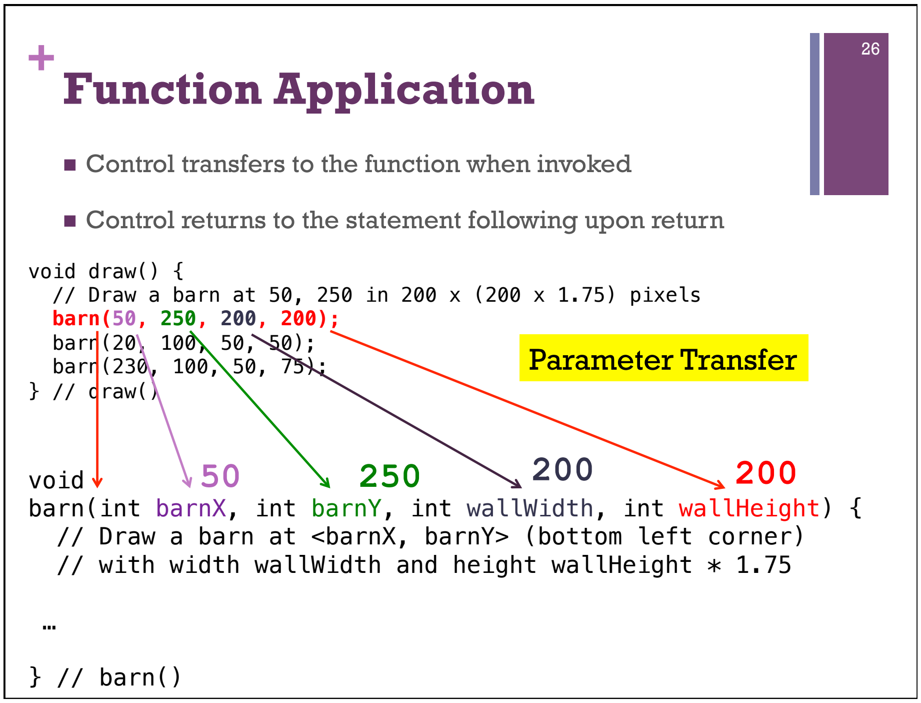
+ Function Application

25

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

```
void setup() {
  // set the size of the canvas
  size(500, 500);
  background(255);

  stroke(128);
  rect(20, 20, 40, 40);
} // setup()
```

```
void size(int newWidth, int newHeight) {
  // set the size of the canvas based on
  // newWidth and newHeight
  width = newWidth;
 …

} // size()
```



+ Function Application

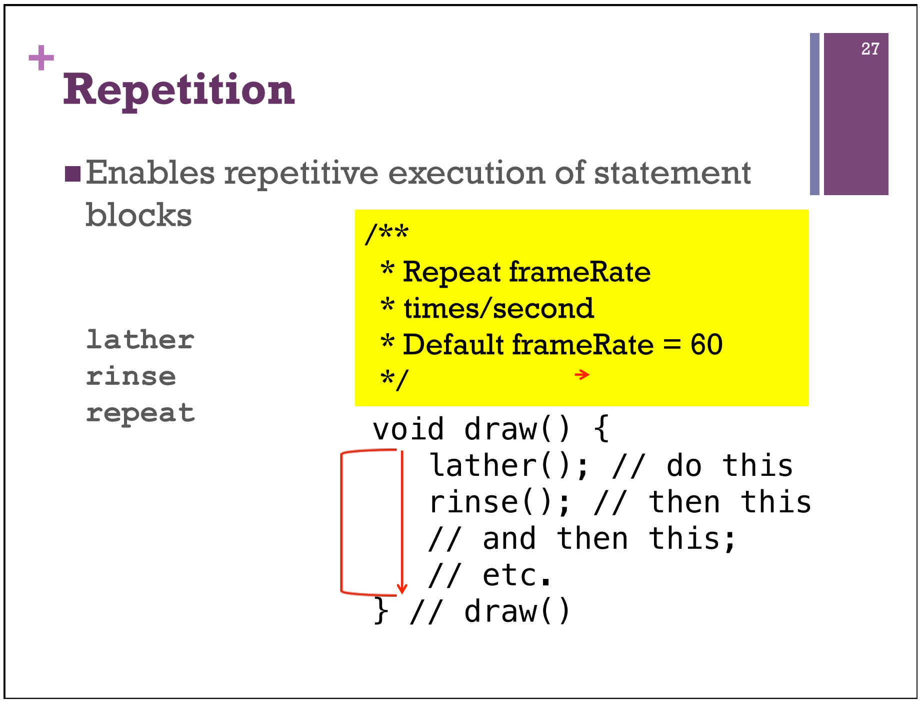
26

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

```
void draw() {
  // Draw a barn at 50, 250 in 200 x (200 x 1.75) pixels
  barn(50, 250, 200, 200);
  barn(20, 100, 50, 50);
  barn(230, 100, 50, 75);
} // draw()
```

Parameter Transfer

```
void          50       250         200         200
barn(int barnX, int barnY, int wallWidth, int wallHeight) {
  // Draw a barn at <barnX, barnY> (bottom left corner)
  // with width wallWidth and height wallHeight * 1.75

 …

} // barn()
```



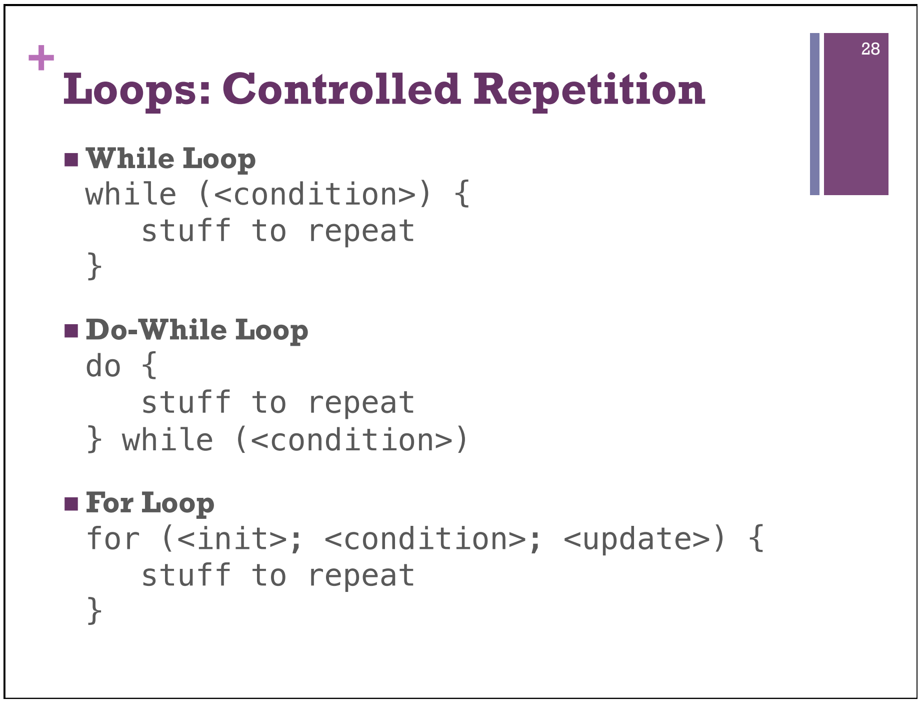
27

Repetition

- Enables repetitive execution of statement blocks

```
lather
rinse
repeat
```

```
/**
 * Repeat frameRate
 * times/second
 * Default frameRate = 60
 */
void draw() {
   lather(); // do this
   rinse(); // then this
   // and then this;
   // etc.
} // draw()
```



28

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
}
```

29

+ 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

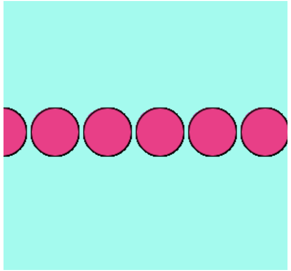
The block
{…}
is called the Loop's Body

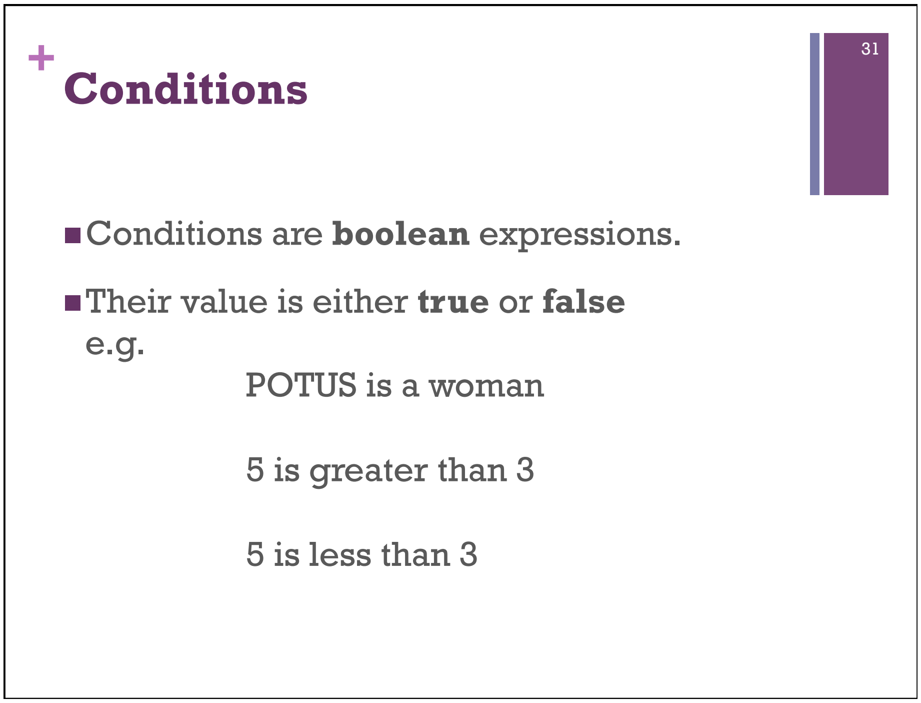
30

+ While Loops

```
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()
```

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





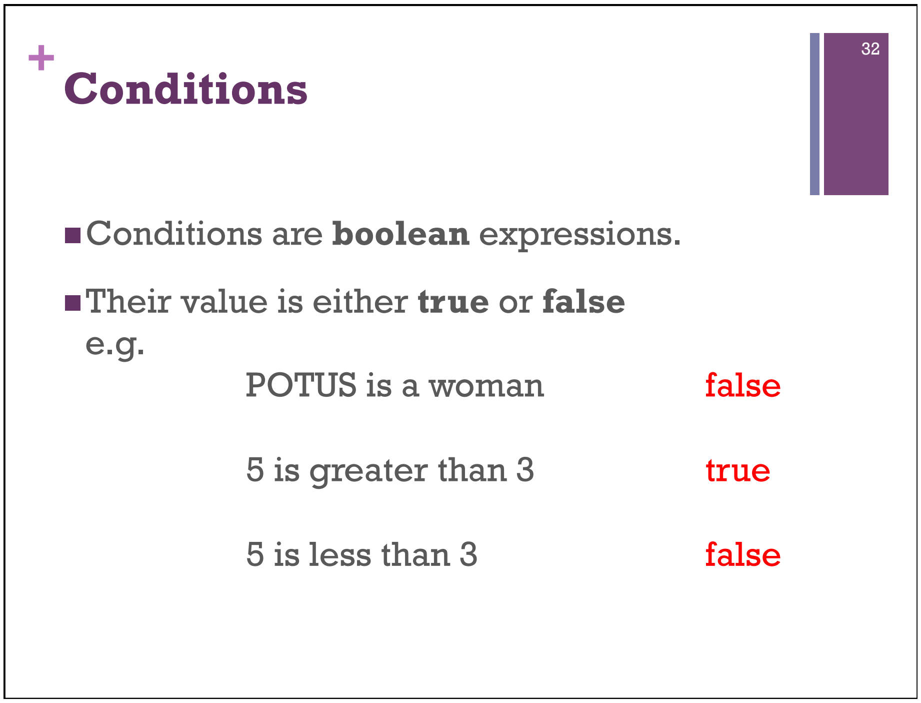
+ Conditions

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

POTUS is a woman

5 is greater than 3

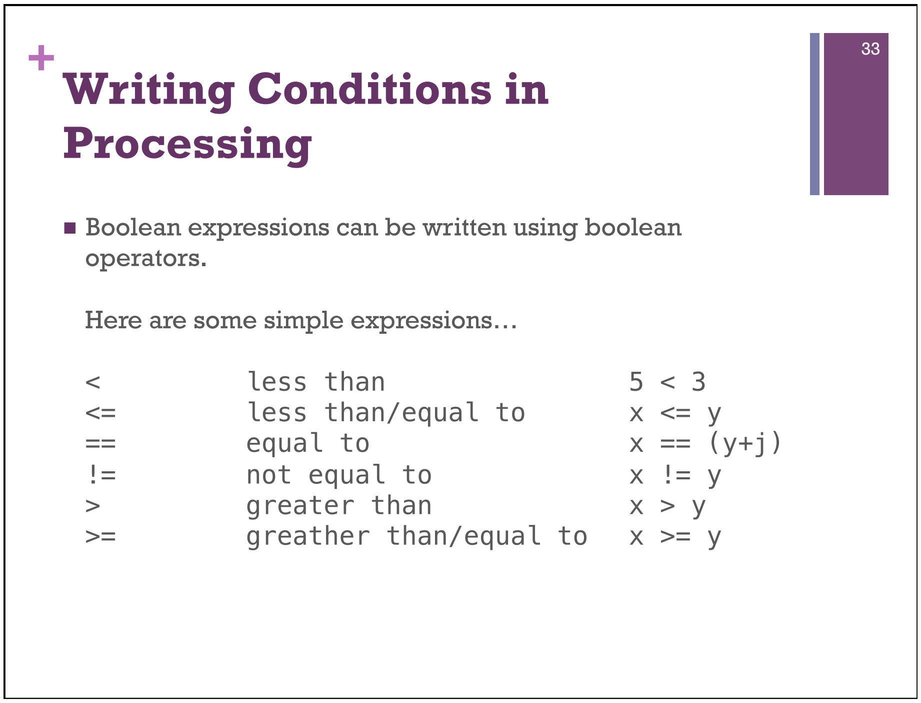
5 is less than 3



+ 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



+

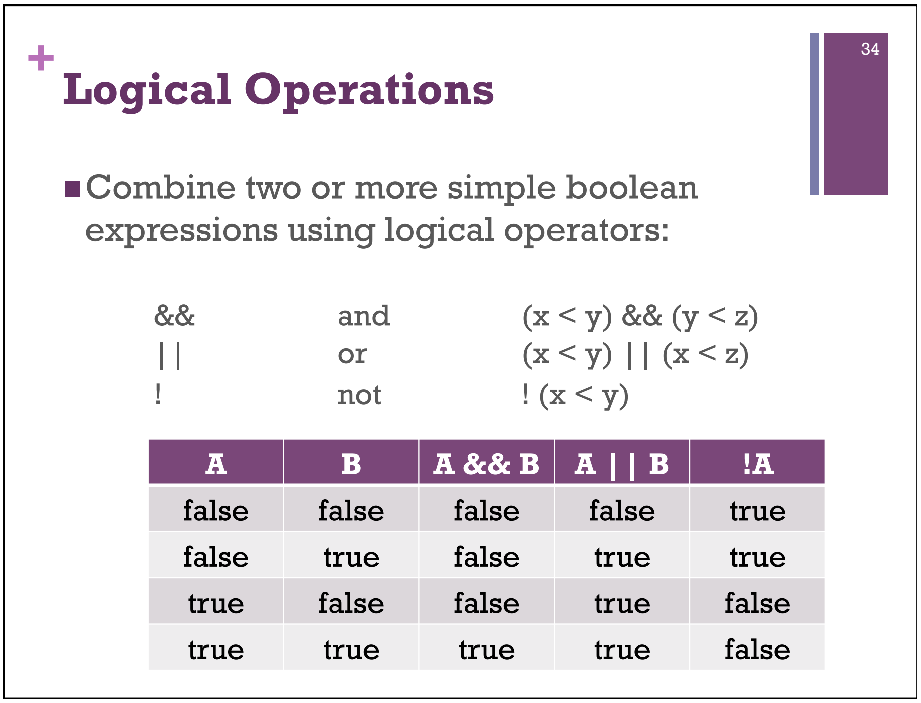
33

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`



+

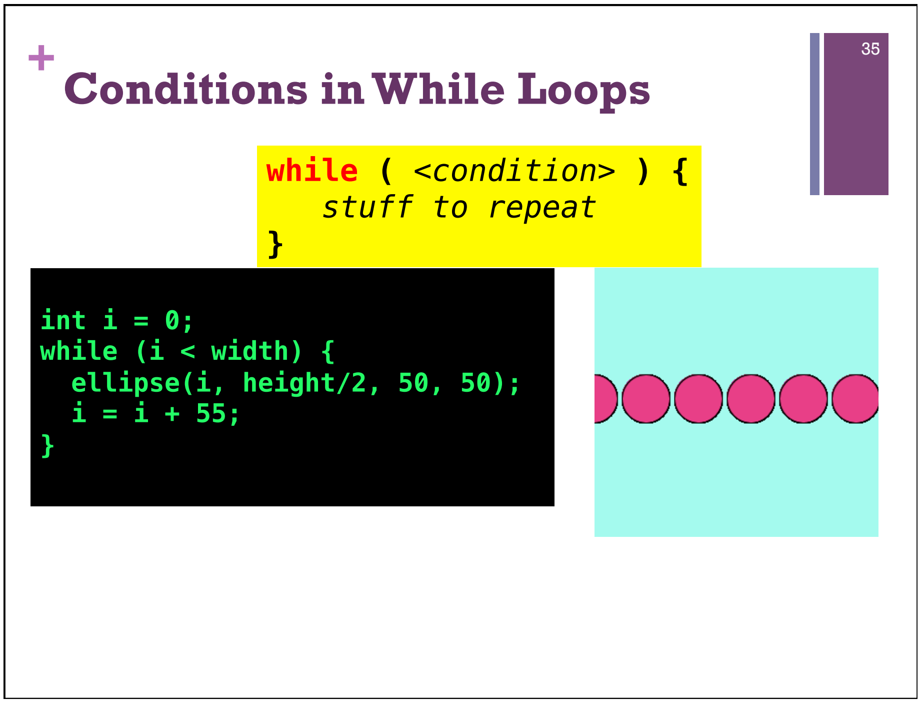
34

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)

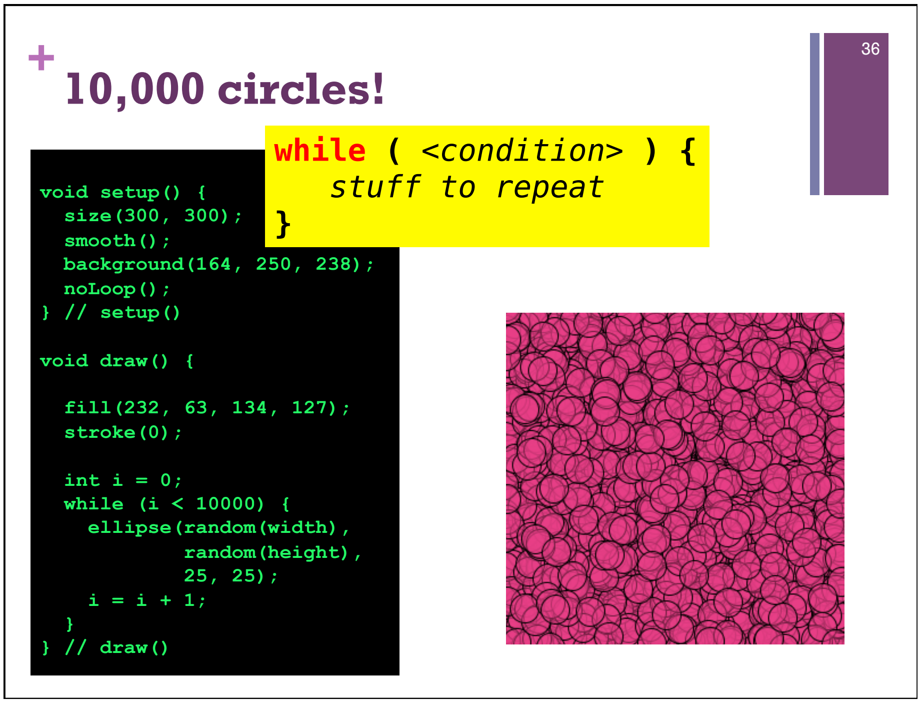
A	B	A && B	A || B	!A
false	false	false	false	true
false	true	false	true	true
true	false	false	true	false
true	true	true	true	false



+ 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()
```