

Processing Boot Camp

Control Structures

Creative Coding & Generative Art in Processing 2
Ira Greenberg, Dianna Xu, Deepak Kumar

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;                               size(200,200);
then do this;                           background(255);
and then do this;
etc.                                     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()
```

The diagram illustrates the control flow between two functions. A red arrow points from the `house(50, 250, 200, 200);` call inside the `draw()` function to the `void house()` definition. A second red arrow points from the closing brace of the `house()` definition back to the line immediately following the `house()` call in the `draw()` function, indicating the return of control.

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
}
```

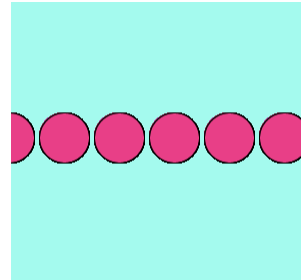
All of these repeat
the stuff in the block

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);  
} // 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	$x \leq y$
==	equal to	$x == (y+j)$
!=	not equal to	$x != y$
>	greater than	$x > y$
>=	greater than/equal to	$x \geq y$

Logical Operations

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

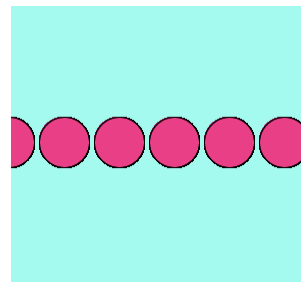
&&	and	(x < y) && (y < z)
	or	(x < y) (x < z)
!	not	!(x < y)

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!

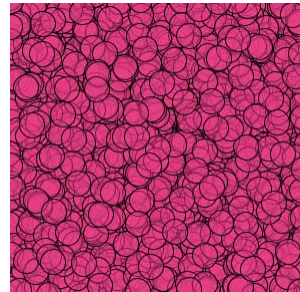
```
while (<condition>){
  stuff to repeat
}
```

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

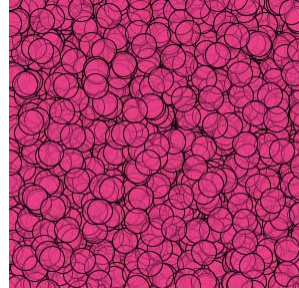
```
void setup() {
  size(300, 300);
  smooth();
  background(164, 250, 238);
} // setup()

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

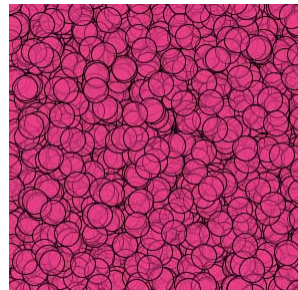
```
for (<init>; <condition>; <update>) {
  stuff to repeat
}
```

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

Loop Initialization

```
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);
    i = i + 1;
}
```

Termination Condition

```
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);
    i = i + 1;
}
```

Loop Update

```
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);
}
```

Loop Body

```
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...