Module 01
Processing Recap
Processing is...

...a language

...a library

...an environment
Variables

A *variable* is a named value. It has a *type* (which can’t change) and a current value (which can change).
Variables

A *declaration* introduces a new variable, and optionally gives it an initial value.

```java
int a;
float b = 6.28;
boolean c = b > 19;
```

Three declarations
Variables

A *declaration* introduces a new variable, and optionally gives it an initial value.

```
int a;
float b = 6.28;
boolean c = b > 19;
```
Variables

A *declaration* introduces a new variable, and optionally gives it an initial value.

```
int a;
float b = 6.28;
boolean c = b > 19;
```
Variables

A *declaration* introduces a new variable, and optionally gives it an initial value.

```java
int a;
float b = 6.28;
boolean c = b > 19;
```
Variables

Say a variable’s name to read from it. Use assignment (\(=\)) to write to it.

Processing includes many built-in names.

• True constants can’t be changed.
• Some variables are meant to be read-only.
• Some are updated automatically, and are meant to be read repeatedly.
Scope

Every declaration in Processing has a scope: the part of the program source code in which that declaration is valid.

Usually either “global” or bounded by the nearest enclosing {}.

Scope is a complicated topic. If in doubt, just avoid re-using the same names!
Control flow

By default, Processing will execute statements in the order they’re given. Control flow can modify that order.
Conditionals

```java
if( keyPressed && key == ' ' ) {
    ellipse( mouseX, mouseY, 20, 20 );
}
```

An if statement
Conditionals

Condition

```cpp
if( keyPressed && key == ' ' ) {
    ellipse( mouseX, mouseY, 20, 20 );
}
```
if( keyPressed && key == ' ' ) {
    ellipse( mouseX, mouseY, 20, 20 );
}
if( keyPressed && key == ' ' ) {
    ellipse( mouseX, mouseY, 20, 20 );
} else {
    rect( mouseX, mouseY, 20, 20 );
}
Conditionals

```cpp
if ( keyPressed ) {
    if ( key == 'e' ) {
        ellipse( mouseX, mouseY, 20, 20 );
    } else if ( key == 'l' ) {
        line( 10, 10, 100, 100 );
    } else {
        rect( mouseX, mouseY, 20, 20 );
    }
}
```
While loops

```c
int y = 0;

while( y < height ) {
    line( 0, y, width, y );
    y = y + 10;
}
```
int y = 0;  // Condition
while (y < height) {
    line(0, y, width, y);
    y = y + 10;
}
While loops

```c
int y = 0;

while (y < height) {
    line(0, y, width, y);
    y = y + 10;
}
```
While loops

```java
int y = 0;

while (y < height) {
    line(0, y, width, y);
    y = y + 10;
}
```

Update!
For loops

```c
for( int y = 0; y < height; y += 10 ) {
    line( 0, y, width, y );
}
```
For loops

Initializer

\[
\text{for} (\text{int } y = 0; y < \text{height}; y += 10 ) \{
    \text{line}(0, y, \text{width}, y);
\}
\]
For loops

for (int y = 0; y < height; y += 10) {
    line(0, y, width, y);
}
For loops

```java
for( int y = 0; y < height; y += 10 ) {
    line( 0, y, width, y );
}
```

Update
For loops

```
for( int y = 0; y < height; y += 10 ) {
    line( 0, y, width, y );
}
```

Body
Functions

A function gives a name to a computation.

Benefits:

• Ease of (error-free) repetition.
• Encapsulation: hide the messy details.
• Abstraction: think about problem solving at a higher level.
• Establish a point of connection between parts of a program.
Calculate the perimeter of a triangle.
Pythagorean theorem

\[ \sqrt{a^2 + b^2} \]
float e1 = sqrt( sq( x2 - x1 ) + sq( y2 - y1 ) );
float e2 = sqrt( sq( x3 - x2 ) + sq( y3 - y2 ) );
float e3 = sqrt( sq( x1 - x3 ) + sq( y1 - y3 ) );
float perim = e1 + e2 + e3;
float measure(float ax, float ay, float bx, float by)
{
    return sqrt(sq(bx - ax) + sq(by - ay));
}
float measure( float ax, float ay, float bx, float by )
{
    return sqrt( sq( bx - ax ) + sq( by - ay ) );
}
float measure( float ax, float ay, float bx, float by )
{
    return sqrt( sq( bx - ax ) + sq( by - ay ) );
}
Parameters

float measure(float ax, float ay, float bx, float by) {
    return sqrt(sq(bx - ax) + sq(by - ay));
}
float measure( float ax, float ay, float bx, float by )
{
    return sqrt( sq( bx - ax ) + sq( by - ay ) );
}
float measure(float ax, float ay, float bx, float by)
{
    return sqrt(sq(bx - ax) + sq(by - ay));
}
float measure( float ax, float ay, float bx, float by )
{
    return sqrt( sq( bx - ax ) + sq( by - ay ) );
}
float e1 = sqrt( sq( x2 - x1 ) + sq( y2 - y1 ) );
float e2 = sqrt( sq( x3 - x2 ) + sq( y3 - y2 ) );
float e3 = sqrt( sq( x1 - x3 ) + sq( y1 - y3 ) );
float perim = e1 + e2 + e3;
float measure( float ax, float ay, float bx, float by )
{
    return sqrt( sq( bx - ax ) + sq( by - ay ) );
}

float e1 = measure( x1, y1, x2, y2 );
float e2 = measure( x2, y2, x3, y3 );
float e3 = measure( x3, y3, x1, y1 );
float perim = e1 + e2 + e3;
float e1 = dist( x1, y1, x2, y2 );
float e2 = dist( x2, y2, x3, y3 );
float e3 = dist( x3, y3, x1, y1 );
float perim = e1 + e2 + e3;
float trianglePerim( 
  float x1, float y1, 
  float x2, float y2, 
  float x3, float y3 )
{
  float e1 = dist( x1, y1, x2, y2 );
  float e2 = dist( x2, y2, x3, y3 );
  float e3 = dist( x3, y3, x1, y1 );
  return e1 + e2 + e3;
}
# Functions

A function takes 0 or more parameters as input and returns 0 or 1 values as output.

<table>
<thead>
<tr>
<th>No return value</th>
<th>0 parameters</th>
<th>Universal command!</th>
<th>1+ parameters</th>
<th>Contingent command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return value</td>
<td>Retrieve hidden information</td>
<td>Calculate something</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- No return value
  - Universal command!
  - Contingent command
- Return value
  - Retrieve hidden information
  - Calculate something
Hooks

Processing knows about a few predetermined function names. If you define functions (*hooks*) with those names, Processing will call them at the right times.

Some libraries add more hooks.
An array is a sequence of values, all of the same type, bundled into a single master value.

```c
float[] temps = {
    -4.8,  -4.79,  -4.764, -4.762,
    -4.764, -4.824, /* 86 more numbers... */
    -1.083, -1.2,  -1.3,  -1.41
};
```
Arrays

An array is a sequence of values, all of the same type, bundled into a single master value.

Array type

```java
float[] temps = {
    -4.8,  -4.79,  -4.764, -4.762,
    -4.764, -4.824, /* 86 more numbers... */
    -1.083, -1.2,  -1.3,  -1.41
};
```
float[] temps = {
    -4.8,   -4.79,   -4.764,  -4.762,
    -4.764, -4.824, /* 86 more numbers... */
    -1.083, -1.2,   -1.3,    -1.41
};

for( int idx = 0; idx < temps.length; ++idx ) {
    if( temps[idx] > 0.0 ) {
        println( "Where's my sunscreen?" );
    }
}
```java
float[] temps = {
    -4.8, -4.79, -4.764, -4.762,
    -4.764, -4.824, /* 86 more numbers... */
    -1.083, -1.2, -1.3, -1.41
};

for( int idx = 0; idx < temps.length; ++idx ) {
    if( temps[idx] > 0.0 ) {
        println( "Where's my sunscreen?" );
    }
}
```
float[] temps = {
    -4.8,  -4.79,  -4.764,  -4.762,
    -4.764, -4.824, /* 86 more numbers... */
    -1.083, -1.2,  -1.3,  -1.41
};

for ( int idx = 0; idx < temps.length; ++idx ) {
    if ( temps[idx] > 0.0 ) {
        println( "Where's my sunscreen?" );
    }
}
Classes and objects

A class introduces a new type. Values of that type (instances) have their own state and behaviour.
class Circle
{
    float cx;
    float cy;
    float radius;

    Circle( float cxIn, float cyIn, float radiusIn )
    {
        cx = cxIn;
        cy = cyIn;
        radius = radiusIn;
    }

    void draw()
    {
        ellipse( cx, cy, 2*radius, 2*radius );
    }
}
class Circle
{
    float cx;
    float cy;
    float radius;

    Circle( float cxIn, float cyIn, float radiusIn )
    {
        cx = cxIn;
        cy = cyIn;
        radius = radiusIn;
    }

    void draw()
    {
        ellipse( cx, cy, 2*radius, 2*radius );
    }
}
class Circle
{
  float cx;
  float cy;
  float radius;

  Circle( float cxIn, float cyIn, float radiusIn )
  {
    cx = cxIn;
    cy = cyIn;
    radius = radiusIn;
  }

  void draw()
  {
    ellipse( cx, cy, 2*radius, 2*radius );
  }
}
class Circle
{
    float cx;
    float cy;
    float radius;

    Circle( float cxIn, float cyIn, float radiusIn )
    {
        cx = cxIn;
        cy = cyIn;
        radius = radiusIn;
    }

    void draw()
    {
        ellipse( cx, cy, 2*radius, 2*radius );
    }
}
class Circle {
    float cx;
    float cy;
    float radius;

    Circle( float cxIn, float cyIn, float radiusIn ) {
        cx = cxIn;
        cy = cyIn;
        radius = radiusIn;
    }

    void draw() {
        ellipse( cx, cy, 2*radius, 2*radius );
    }
}
Circle[] circs;

void setup()
{
    circs = new Circle[10];
    for ( int idx = 0; idx < circs.length; ++idx ) {
        circs[idx] = new Circle(
            random(100), random(100), random(20) );
    }
}

void draw()
{
    background( 255 );
    for ( int idx = 0; idx < circs.length; ++idx ) {
        circs[idx].draw();
    }
}
null

null is a special keyword that represents a non-existent value for every class. It is the default value for variables of class type. It’s illegal to access any fields or methods of null.

```java
void draw()
{
    background( 255 );
    for ( int idx = 0; idx < circs.length; ++idx ) {
        if ( circs[idx] != null ) {
            circs[idx].draw();
        }
    }
}
```
```java
void getIntersection( Line l1, Line l2 )
{
    if( ... ) {
        return new Point( ..., ... );
    } else {
        return null;
    }
}
```
What’s this?

this is a keyword that can be used inside the body of a method. It always refers to “the object that received the method call”. We won’t need it ourselves, but we’ll see it sometimes when using libraries.

```java
import processing.video.*;

Capture cam;

void setup()
{
    cam = new Capture( this, 320, 240 );
    cam.start();
}
```
What’s this?

This is a keyword that can be used inside the body of a method. It always refers to “the object that received the method call”. We won’t need it ourselves, but we’ll see it sometimes when using libraries.

```java
import processing.video.*;

Capture cam;

void setup()
{
  cam = new Capture(this, 320, 240);
  cam.start();
}
```

Jacques Carelman, Cocepot for Masochists