Module 01

Processing Recap

CS 106 Winter 2018

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.

```
    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;
```

Type

Variables

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

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

Name

Variables

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

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

Initial value

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.

CQ

What does this program print?

```
int a = 17;

void test( int a )
{
  println( a + 5 );
}

void setup()
{
  func( 10 );
}
```

(A) 5

(B) 15

(C) 22

(D) a + 5

(E) Nothing

Scope

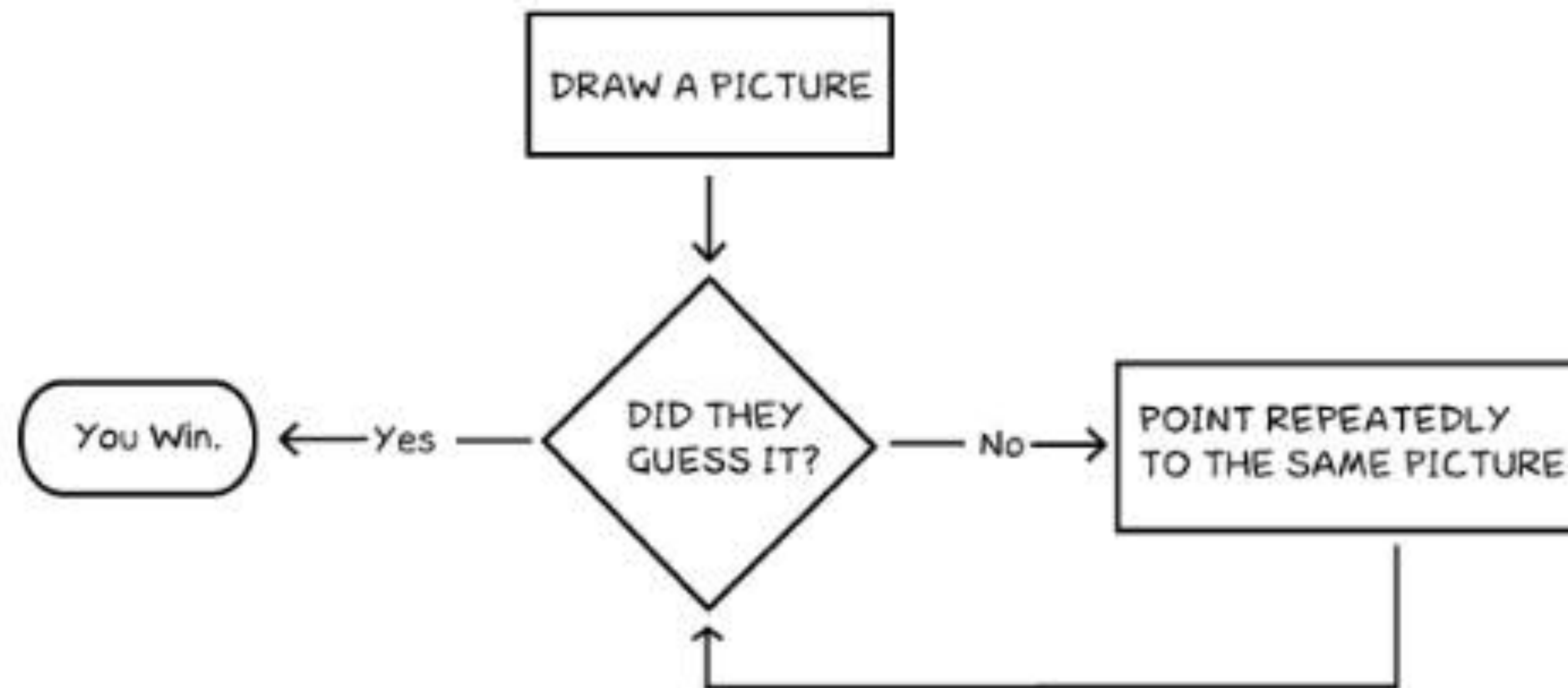
Every declaration in Processing has a *scope*: the part of the program source code in which that declaration is valid.

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

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

An if statement

Conditionals

Condition

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

Conditionals

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

Body

Conditionals

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

Conditionals

```
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

```
int y = 0;

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

While loops

```
int y = 0;

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

Condition

While loops

```
int y = 0;

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

Body

While loops

```
int y = 0;

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

Update!

For loops

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

For loops

Initializer

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

For loops

Condition

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

For loops

Update

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

For loops

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

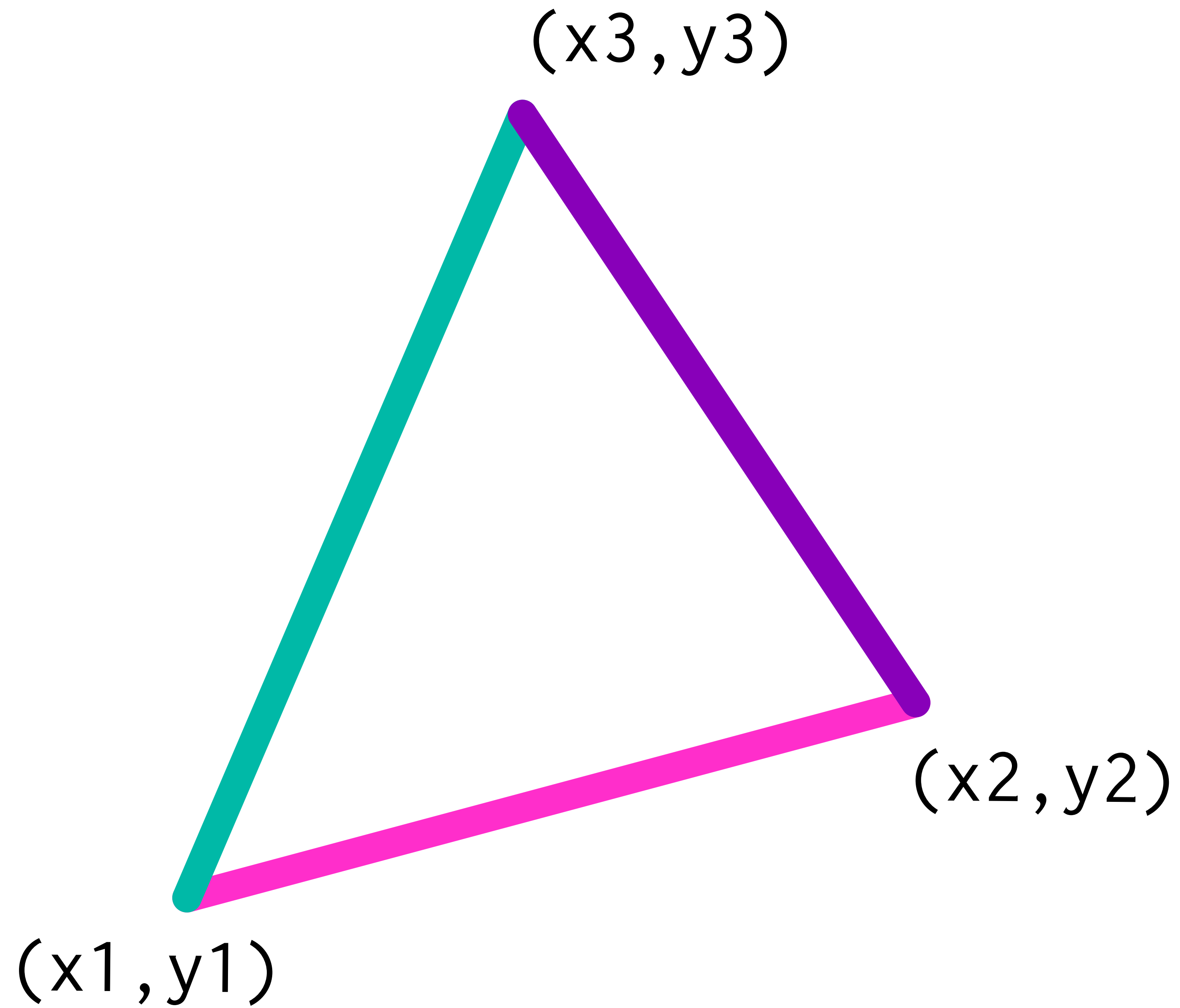
Body

Functions

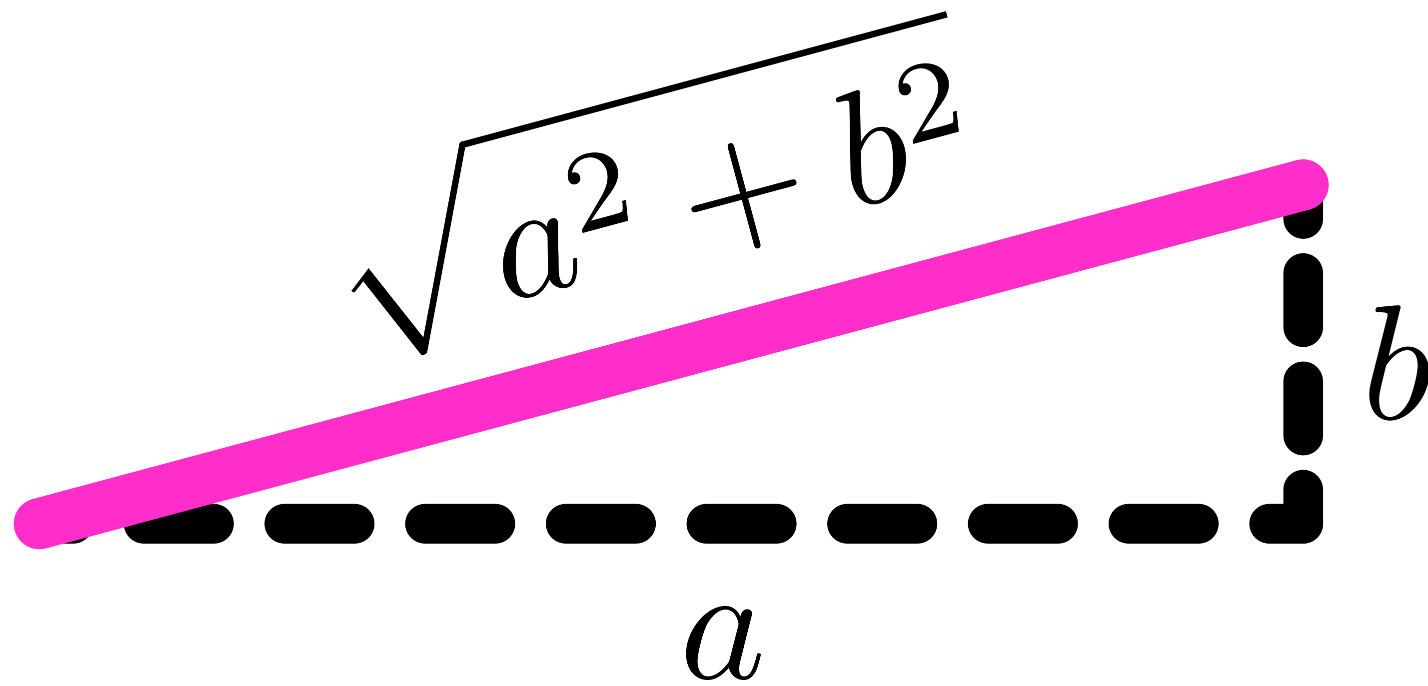
A *function* gives a name to a computation.

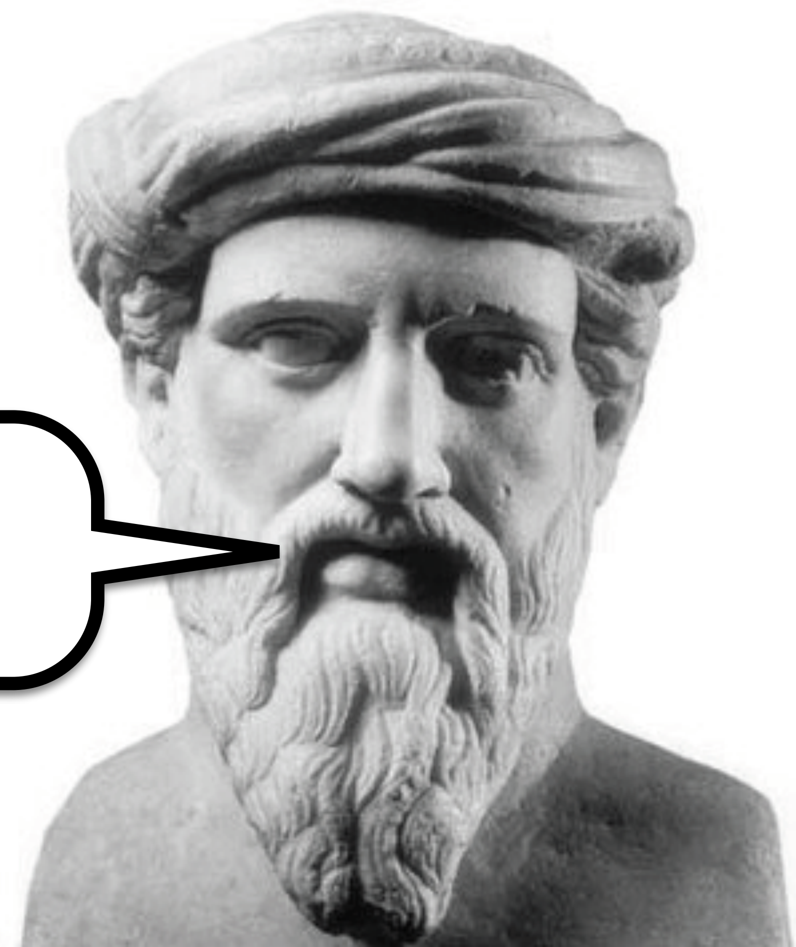
Benefits:

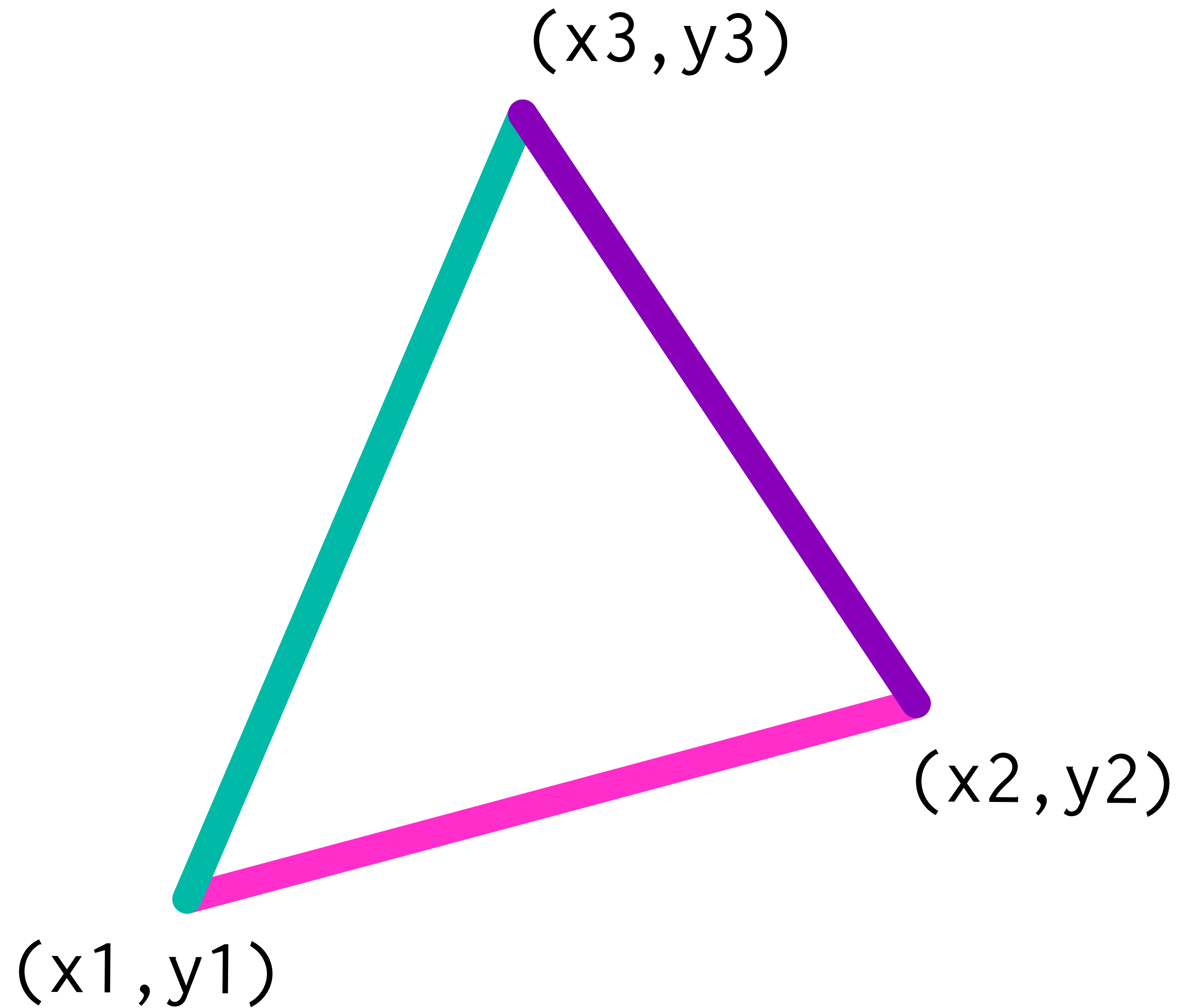
- Ease of (error-free) repetition.
- Encapsulation: hide the messy details.
- Abstraction: think about problem solving at a higher level.
- Establish a point of connection between parts of a program.



Calculate the perimeter of a triangle.


$\sqrt{a^2 + b^2}$
b
a


Pythagorean theorem




```
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 ) );
}
```

Return type

```
float measure( float ax, float ay, float bx, float by )
{
  return sqrt( sq( bx - ax ) + sq( by - ay ) );
}
```

Function name

```
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 ) );
}
```

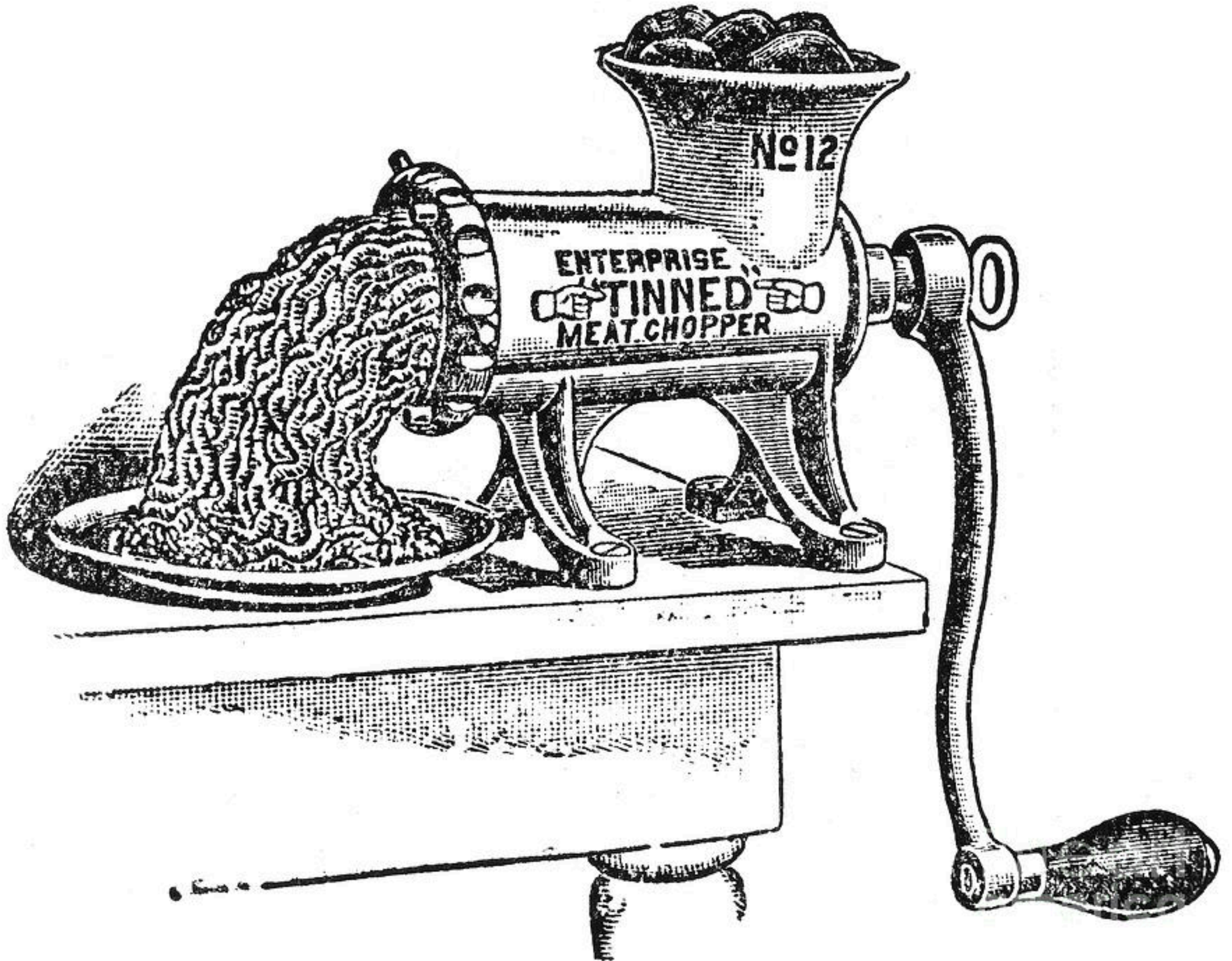
```
float measure( float ax, float ay, float bx, float by )
{
  return sqrt( sq( bx - ax ) + sq( by - ay ) );
}
```

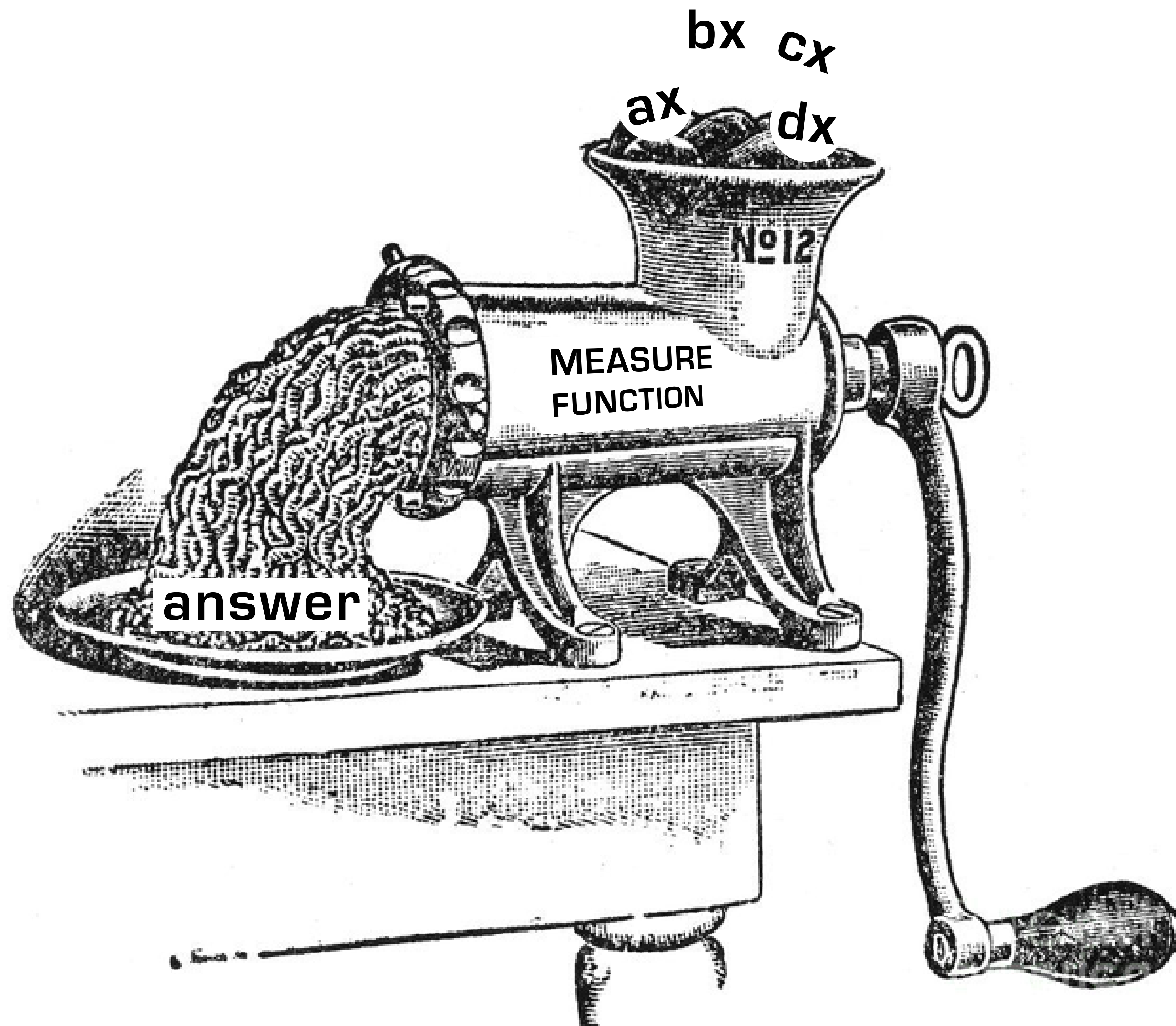
Body

```
float measure( float ax, float ay, float bx, float by )
{
  return sqrt( sq( bx - ax ) + sq( by - ay ) );
}
```

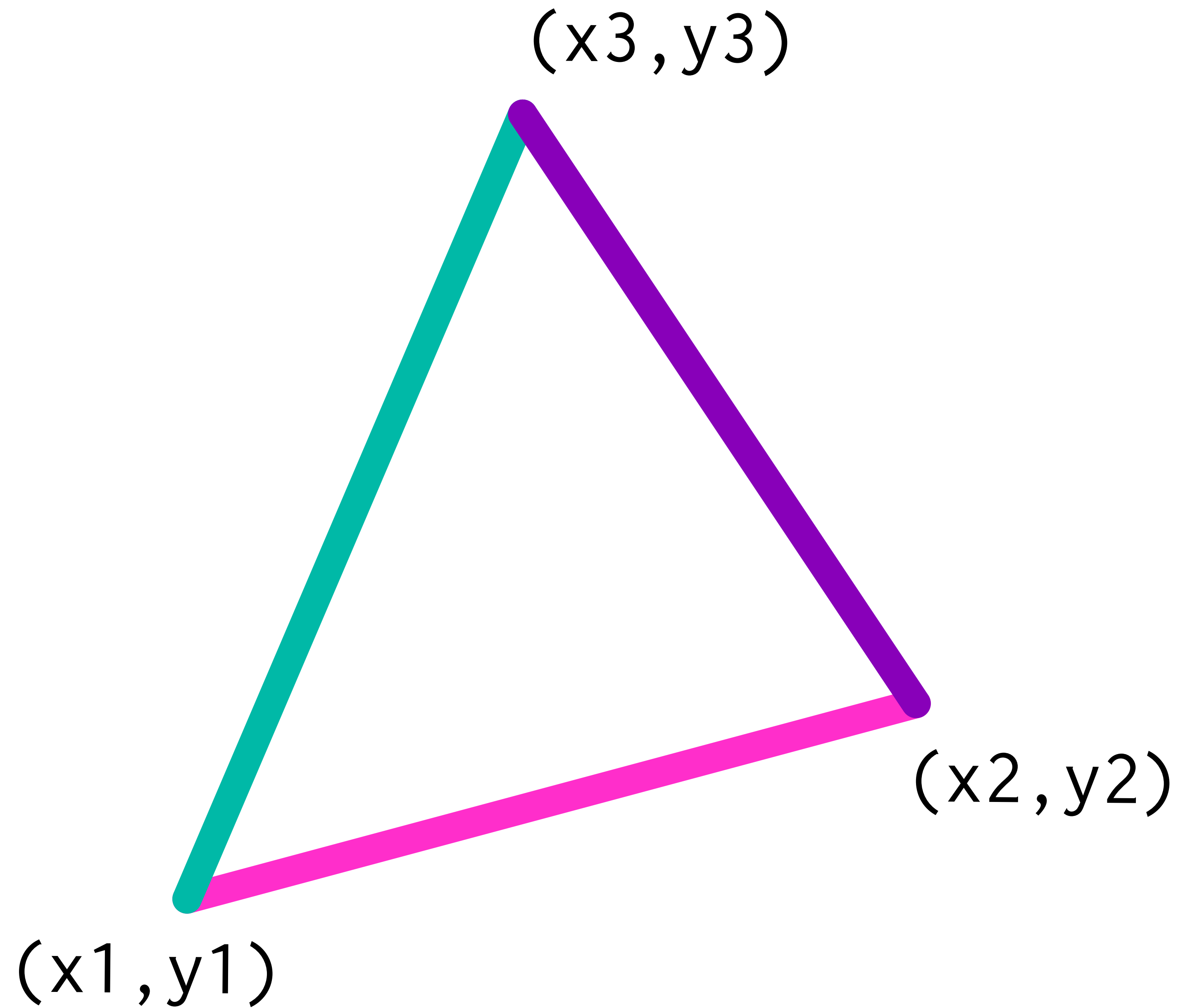
Return statement

If a function's return type is not `void`, it has to contain one or more `return` statements.

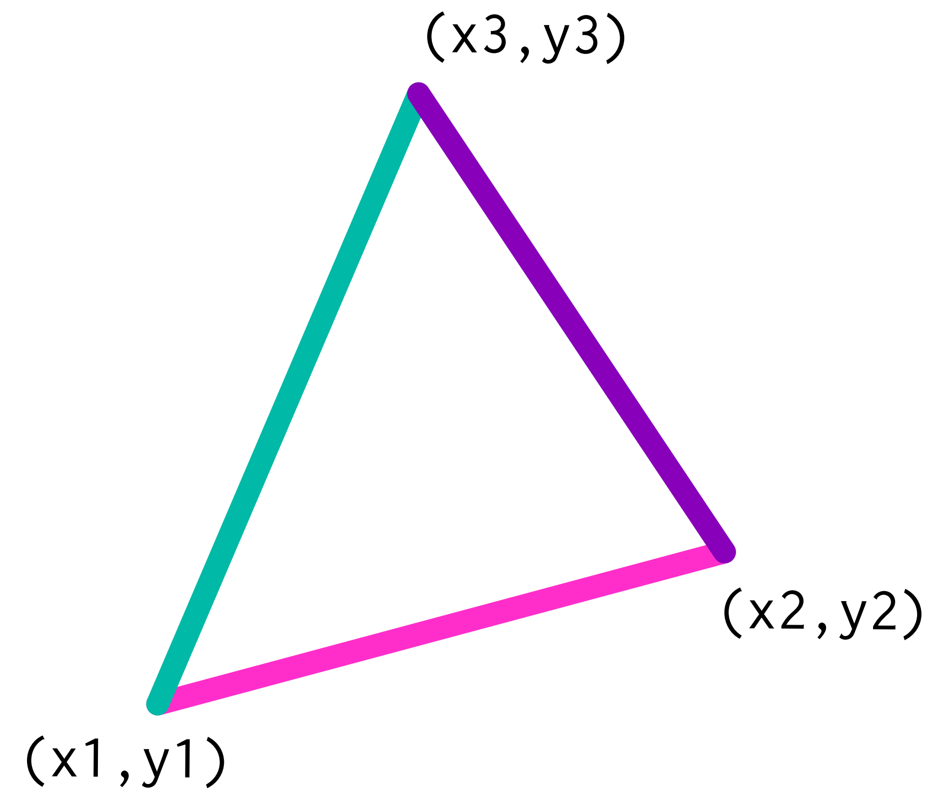

№ 12
ENTERPRISE
"TINNED"
MEAT CHOPPER



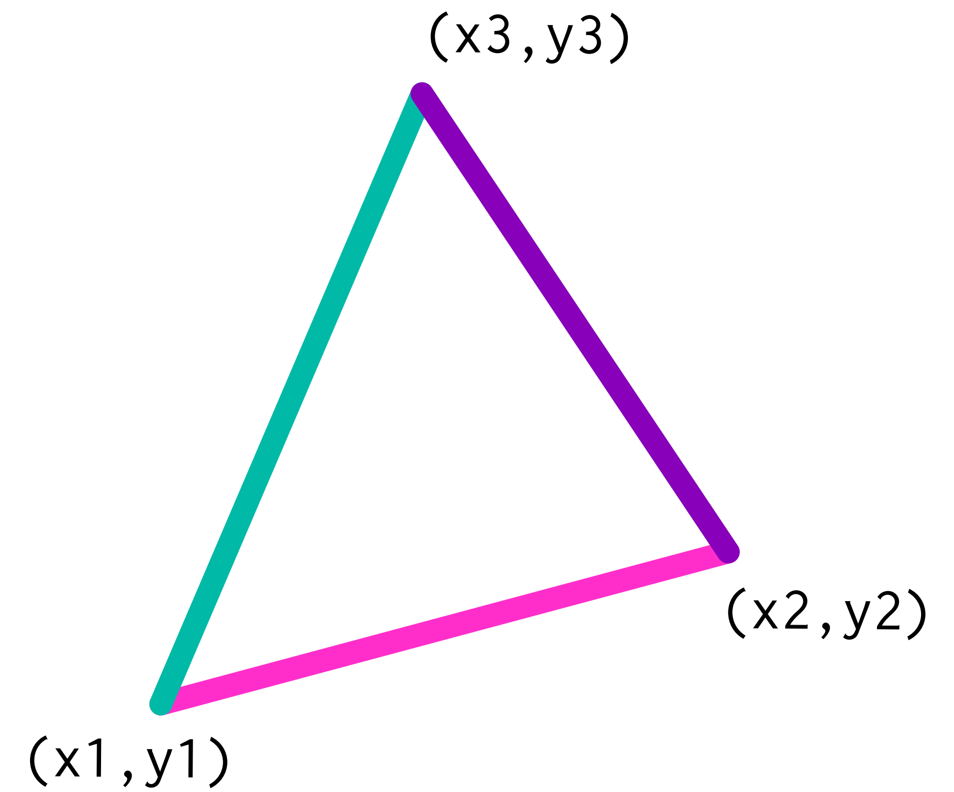

```
float measure( float ax, float ay, float bx, float by )
{
  return sqrt( sq( bx - ax ) + sq( by - ay ) );
}
```




```
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;
```

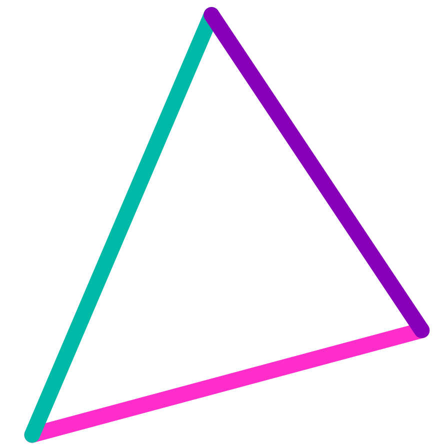


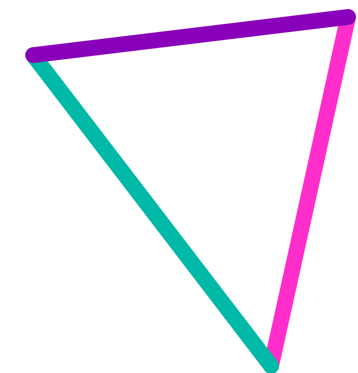

```
float measure( float ax, float ay, float bx, float by )
{
  return sqrt( sq( bx - ax ) + sq( by - ay ) );
}

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;
```

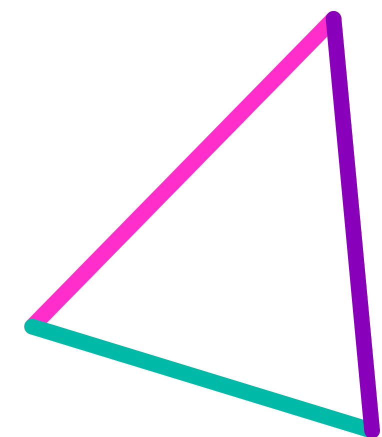


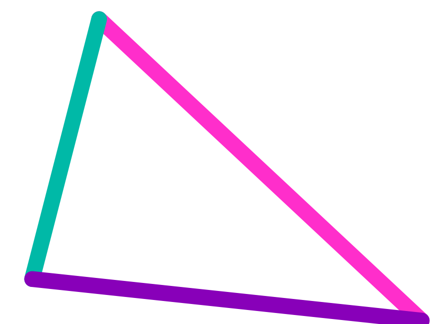

```
float e1 = dist( x1, y1, x2, y2 );
float e2 = dist( x2, y2, x3, y3 );
float e3 = dist( x3, y3, x1, y1 );
float perim = e1 + e2 + e3;
```

```
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.

	0 parameters	1+ parameters
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.

Examples: `setup()`, `draw()`, `mousePressed()`, `keyPressed()`

Some libraries add more hooks.

Arrays

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

```
float[] ts1 = {
  -4.8,   -4.79,  -4.764, -4.762,
  -4.764, -4.824, /* 86 more numbers... */
  -1.083, -1.2,   -1.3,   -1.41
};

float[] ts2 = {
  -21.08933, -21.814,  -22.542, -22.01667,
  -20.912,   -21.564 /* 86 more numbers... */
  -27.48999, -27.43200, -27.88466, -28.09467
};
```

Arrays

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

Array type

```
float[] ts1 = {
   -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.

Initial value(s)

```
float[] ts1 = {
  -4.8,   -4.79,  -4.764, -4.762,
  -4.764, -4.824, /* 86 more numbers... */
  -1.083, -1.2,   -1.3,   -1.41
};
```

```
float[] ts1 = {
  -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 < ts1.length; ++idx ) {
  if( ts1[idx] > 0.0 ) {
    println( "Where's my sunscreen?" );
  }
}
```

```
float[] ts1 = {
  -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 < ts1.length; ++idx ) {
  if( ts1[idx] > 0.0 ) {
    println( "Where's my sunscreen?" );
  }
}
```

Array size

```
float[] ts1 = {
  -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 < ts1.length; ++idx ) {
  if( ts1[idx] > 0.0 ) {
    println( "Where's my sunscreen?" );
  }
}
```

Element access

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  Type name
{
  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 );
  }
}
```

Fields **(per-instance state)**

```
class Circle
{
  float cx;
  float cy;
  float radius;
```

***Constructor* (initialize state)**

```
  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 );
  }
}
```

***Method* (behaviour)**

```
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`**.**

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

```
Point getIntersection( Line l1, Line l2 )
{
  if( ... ) {
    return new Point( ..., ... );
  } else {
    return null;
  }
}
```

What's this?

In CS 106, you can think of the keyword `this` as meaning "this sketch". Some external libraries want to know which sketch they're running in when you initialize them.

```
import processing.video.*;

Capture cam;

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

The keyword `this` is actually a general feature of many object-oriented languages. It refers to the instance of a class that received a method call. It's relevant in Processing because the entire sketch is actually one big class, even though you never see that.

The fact that you have to use `this` explicitly is a design flaw in Processing.

Jacques Carelman, *Coffeepot for Masochists*