OOP:

Object-Oriented Programming

OOP allows us to create **classes** that define **objects**. The class is the template from which instances of the class are created - an object.

Objects can their own have variables and functions.

The object's class defines which variables and functions it has and the code for those functions.

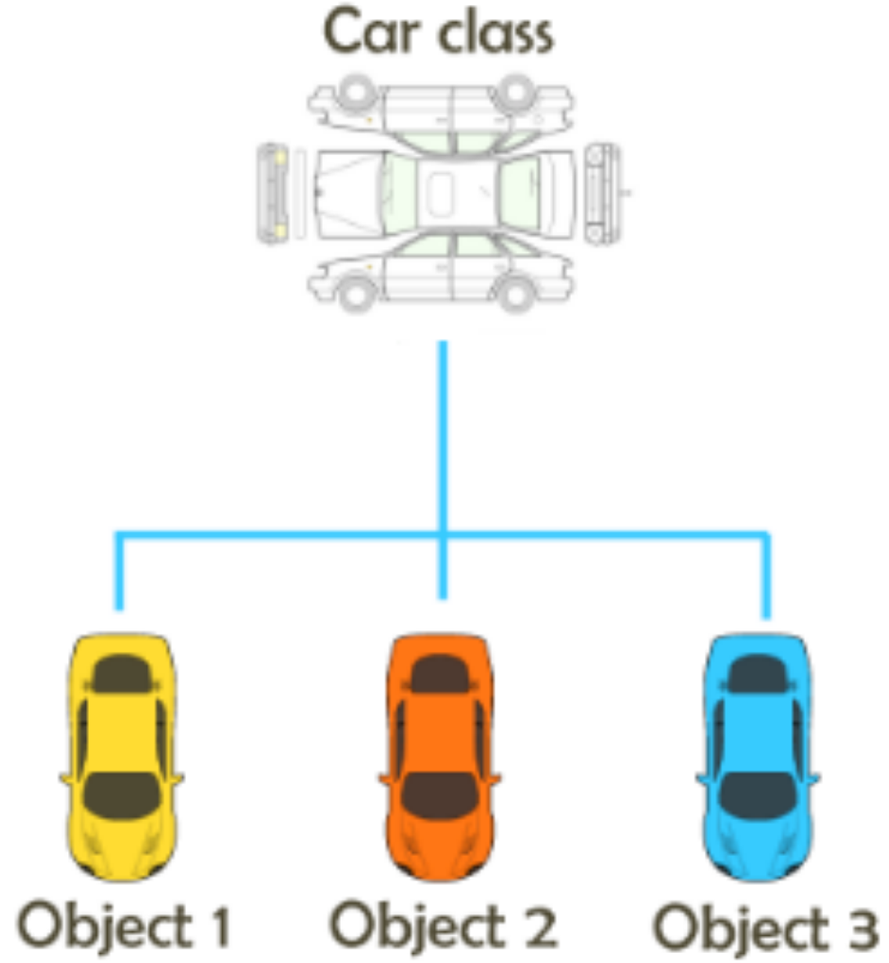
Cookie-cutter Metaphor

Class is the cookie-cutter

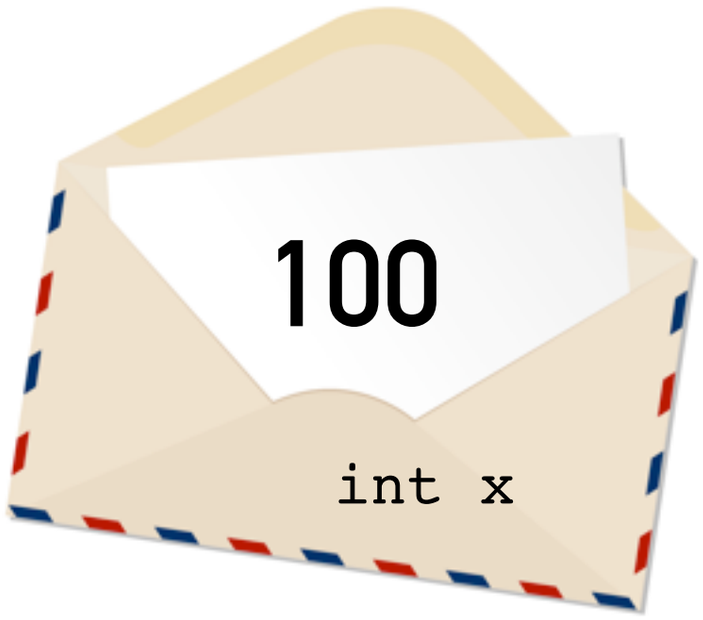
Objects are the cookies made from a cookie-cutter

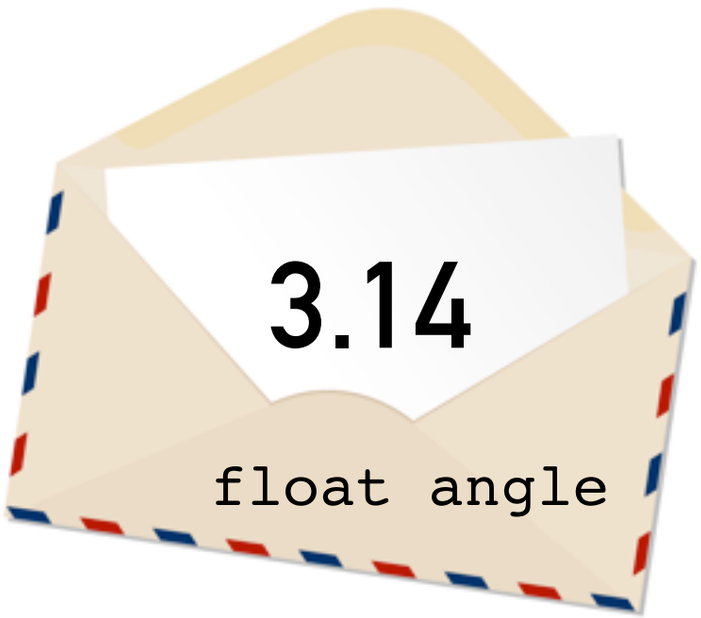
When an object is created it gets all its own variables that the class defined it will have and can execute any of the functions in the class.

Each object has the same template but the value of its variables can change.



Primitive Data Type
(holds single value: int, float, boolean etc)





Objects
(complex data type that can hold
multiple variables and functions)

Writing a class template in Processing:

```
class Particle{

    float x;
    float y;

    Particle(){
      x = 0;
      y = 0;
    }

    void update(){
      x += 1;
      y += 1;
    }
}
```

name of the class (can be anything but convention is to capitalize first letter)

```
class Particle{

    float x;
    float y;

    Particle(){
      x = 0;
      y = 0;
    }

    void update(){
      x += 1;
      y += 1;
    }
}
```

variables that belong to this class. all objects of the class will have these variables

constructor - called when an object is initialized. variables are often given initial values here

a function of the class. class functions can have any name and each object of the class can call the functions

Declaring and initializing an object of a class:

here ***p*** is the name of the object (this can be named anything we want) and ***Particle*** is its class type (this must be exactly came as in the class)

```
Particle p;

void setup(){
    p = new Particle();
}
```

Accessing an object's variable:

use a period after the object name to refer to its internal variables

```
Particle p;

void setup(){

    p = new Particle();

}

void draw(){

  ellipse( p.x, p.y, 20, 20);

}
```

Using an object's function:

also use a period after the object name to call its internal functions

```
Particle p;

void setup(){

   p = new Particle();

}

void draw(){

  p.update();

  ellipse( p.x, p.y, 20, 20);

}
```

Arrays of objects

Declare array

this will initialize an array of 100 objects of the class type Particle, but it does not initialize each object in that array

```
Particle [] dots = new Particle[100];
```

Initialize each object in array:

loop through each object in the array and initialize it with ***new***

```
Particle [] dots = new Particle[100];

void setup(){
    for( int i = 0; i < dots.length; ++){
      dots[i] = new Particle();
    }

}
```

```
Particle [] dots = new Particle[100];

void setup(){

   for( int i = 0; i < dots.length; ++){
     dots[i] = new Particle();
   }

}

void draw(){

   for( int i = 0; i < dots.length; ++){
     dots[i].update();
   }

}
```

ArrayLists

ArrayLists

An ArrayList is like a special type of array that can grow and shrink

ArrayLists can only hold objects (not primitive data types like int, float, byte - there are other structures for this)

Declaring an ArrayList:

```
ArrayList<Particle> particles;
```

declare we are making an *ArrayList*

give the ArrayList a name
(can be anything)

```
ArrayList<Particle> particles;
```

tell what *type* of objects
will be in the ArrayList

Initialize an ArrayList:

```
ArrayList<Particle> particles;

void setup(){

  size(300,300);

  particles = new ArrayList<Particle>();
}
```

Add an object to the ArrayList with add:

```
ArrayList<Particle> particles;

void setup(){

  size(300,300);

  particles = new ArrayList<Particle>();

  particles.add( new Particle() );
}
```

Access object in the ArrayList with get:

```
void draw(){

  for( int i = 0; i < particles.size(); i++){
    Particle p = particles.get(i);
    p.update();
    p.display();
  }

}
```

Note: loops backwards!

```
void draw(){

  for( int i = 0; i < particles.size(); i++){
    Particle p = particles.get(i);
    p.update();
    p.display();
  }

  for( int i = particles.size()-1; i > 0; i--){
    Particle p = particles.get(i);
    if(p.x > width){
      particles.remove(i);
    }
  }

}
```