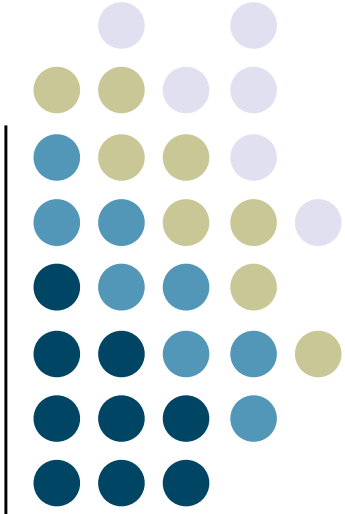
Object-Oriented Programming in Processing



Georgia Tech

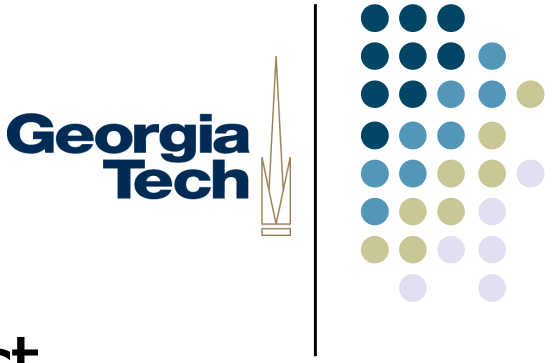
Object-Oriented Programming

- We've (kinda) been doing this since Day 1:
  - Python is a deeply object oriented language
  - Most of the data types we were using (strings, list, dictionaries) were **objects**
- Now, going to shift gears to exploring more about what objects are, how/when to use them
- Explore examples in **Processing**
- Much of this knowledge will be transferrable when we start Java

"Objects"

- Objects are simply data+functionality, in a self-contained unit
- In programming terms, objects have
  - variables associated with them (sometimes called instance variables or member data)
  - functions associated with them (called methods)

"Classes"

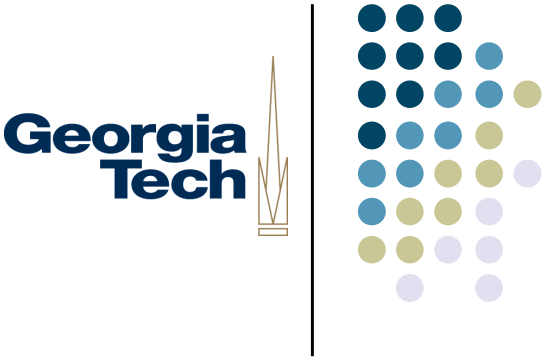


- Classes are the blueprints for an object
  - Think of them like a cookie cutter or a template
  - Can create multiple copies from the pattern
- Each object that is created based on a class's blueprint is called an **instance** of that class
- (Classes are like the abstract description of the thing that becomes an object)
  - Class = cookie cutter
  - Object/instance = cookie

Example

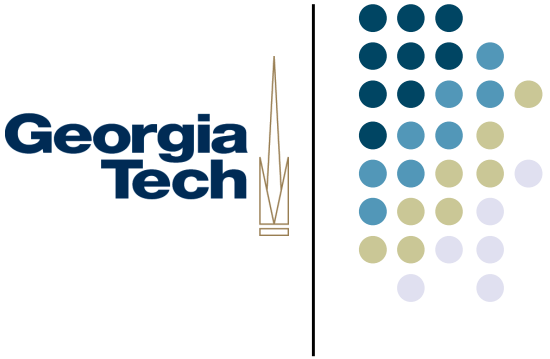
- Let's say we want to write a program that moves a car horizontally across the screen
  - (We'll just use a simple rectangle for the car to make our job easier)
- What information will we need to keep track of for our "car"?
  - color, location (x & y), speed
- What functionality pertains to our "car"?
  - Draw the car on the screen, change its location based on its speed, initialize color/location/etc.

Doing it without OO (pseudocode)



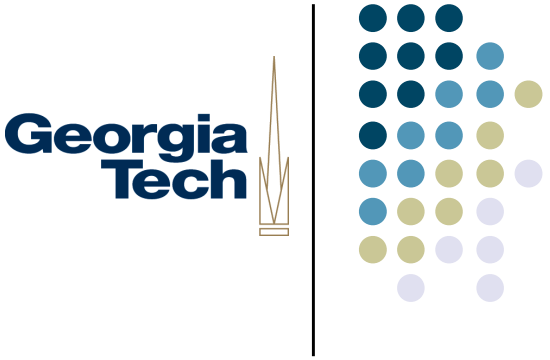
- Global variables
  - Car color
  - Car X & Y location
  - Car speed
- Setup code:
  - Initialize car color
  - Initialize car location to starting point
  - Initialize car speed
- Draw code:
  - Fill background
  - Display car at current location, with current color
  - Increment car's location by speed

Doing it without OO



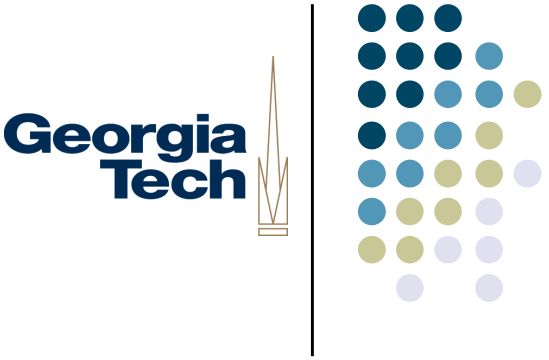

```
color c = color(0);
float x = 0;
float y = 100;
float speed = 1;

void setup() {
  size(200,200);
}

void draw() {
  background(255);
  move();
  display();
}

void move() {
  x = x + speed;
  if (x > width) {
    x = 0;
  }
}

void display() {
  fill(c);
  rect(x,y,30,10);
}
```

That's a lot of stuff spread all over our program!



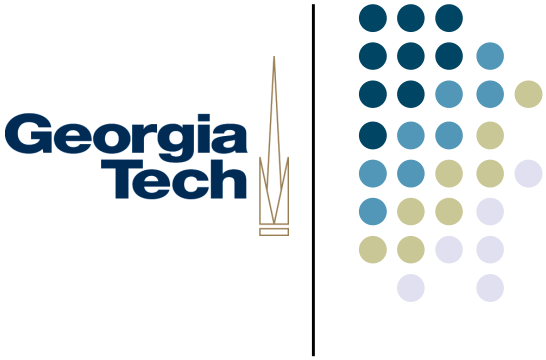
- In an OO development style:
  - Take all of the variables and functions out of the main program
  - Put them inside a car object
  - Car object keeps track of its data
    - Color, location, speed
  - Also handles "the stuff it can do" via methods
    - Display it, drive it, etc.

Pseudocode for the new version



- Global variables in our program:
  - The car object
- Setup code:
  - Initialize the car object
- Draw code:
  - Fill the background
  - Display the car object
  - Drive the car object

Code for the new version




```
Car myCar;

void setup()  {
  myCar = new Car();
}

void draw()  {
  background(255);
  myCar.drive();
  myCar.display();
}
```

- Simpler, and less to keep track of

Defining the class

- Before we can create a Car object, we have to define its class!
  - Write the "cookie cutter"

- All classes have four parts to them:
  - The class's name
  - The data associated with the class (variables)
  - A "constructor" function (run at initialization time)
  - The functions associated with the class (its methods)

```
// Simple non OOP Car
```

```
color c;
float xpos;
float ypos;
float xspeed;
```

```
void setup() {
  size(200,200);
  c = color(255);
  xpos = width/2;
  ypos = height/2;
  xspeed = 1;
}
```

```
void draw() {
  background(0);
  display();
  drive();
}
```

```
void display () {
  rectMode(CENTER);
  fill(c);
  rect(xpos,ypos,20,10);
}

void drive() {
  xpos = xpos + xspeed;
  if (xpos > width) {
    xpos = 0;
  }
}
```

→

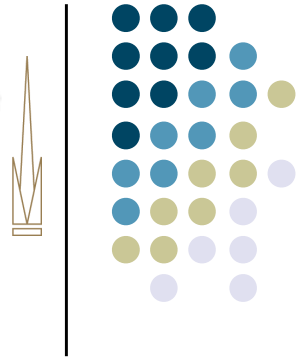
```
class Car {
  color c;
  float xpos;
  float ypos;
  float xspeed;

  Car() {
    c = color(255);
    xpos = width/2;
    ypos = height/2;
    xspeed = 1;
  }

  void display() {
    rectMode(CENTER);
    fill(c);
    rect(xpos,ypos,20,10);
  }

  void drive() {
    xpos = xpos + xspeed;
    if (xpos > width) {
      xpos = 0;
    }
  }
}
```

→ The class name

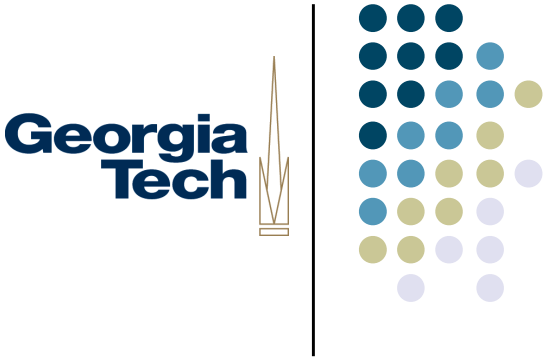
→ Data

→ Constructor

→ Functionality



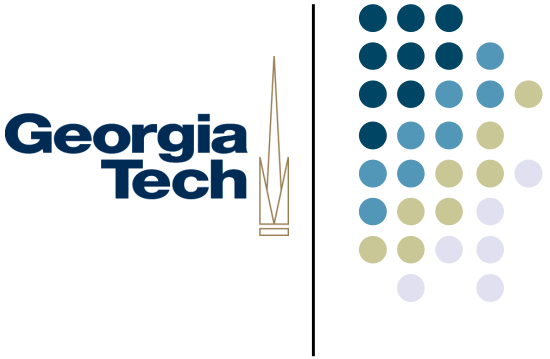
Declaring and Using Objects



- Declaring a normal variable:
  - int var;
- Declaring an object:
  - Car car;

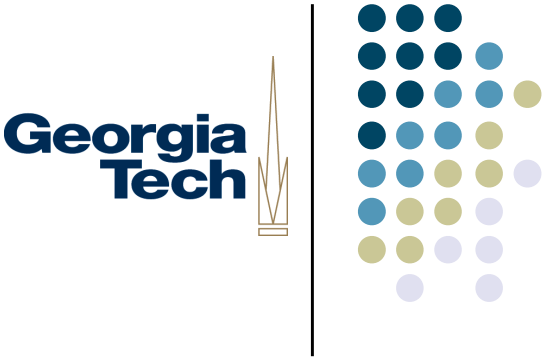
- Initializing a normal variable:
  - var = 10;
- Initializing an object:
  - car = new Car(); // use the "new" operator
  - // this automatically runs the constructor

Putting it all together…



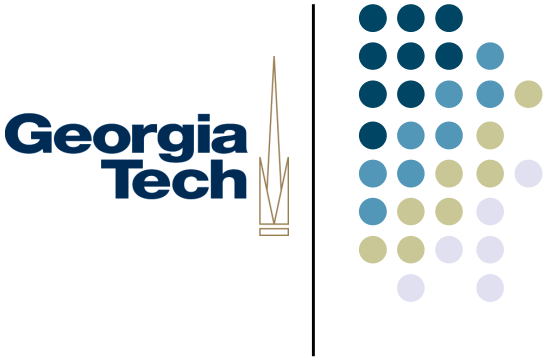
```
// Class definition
class Car {
  color c;
  float xpos;
  float ypos;
  float xspeed;

  Car() {
    c = color(0);
    xpos = width/2;
    ypos = height/2;
    xspeed = 1;
  }

  void display() {
    rectMode(CENTER);
    fill(c);
    rect(xpos, ypos, 20, 10);
  }

  void drive() {
      xpos = xpos + xspeed;
      if (xpos > width) {
        xpos = 0;
      }
    }
}

// Main code
Car myCar;

void setup() {
  size(200,200);
  myCar = new Car();
}

void draw() {
  background(255);
  myCar.drive();
  myCar.display();
}
```

Haven't we just moved the code around?



- Well… yeah.
- But this version is still better:
  - Once we get Car working, we can forget about it
    - CS concept: "encapsulation" - treat the code as a "black box" without having to worry about how it works
    - Each instance is isolated from every other instance
  - We can add new cars with only minor changes to the code!
    - Think about what a mess this would be with our original, non-OO version

Multiple cars



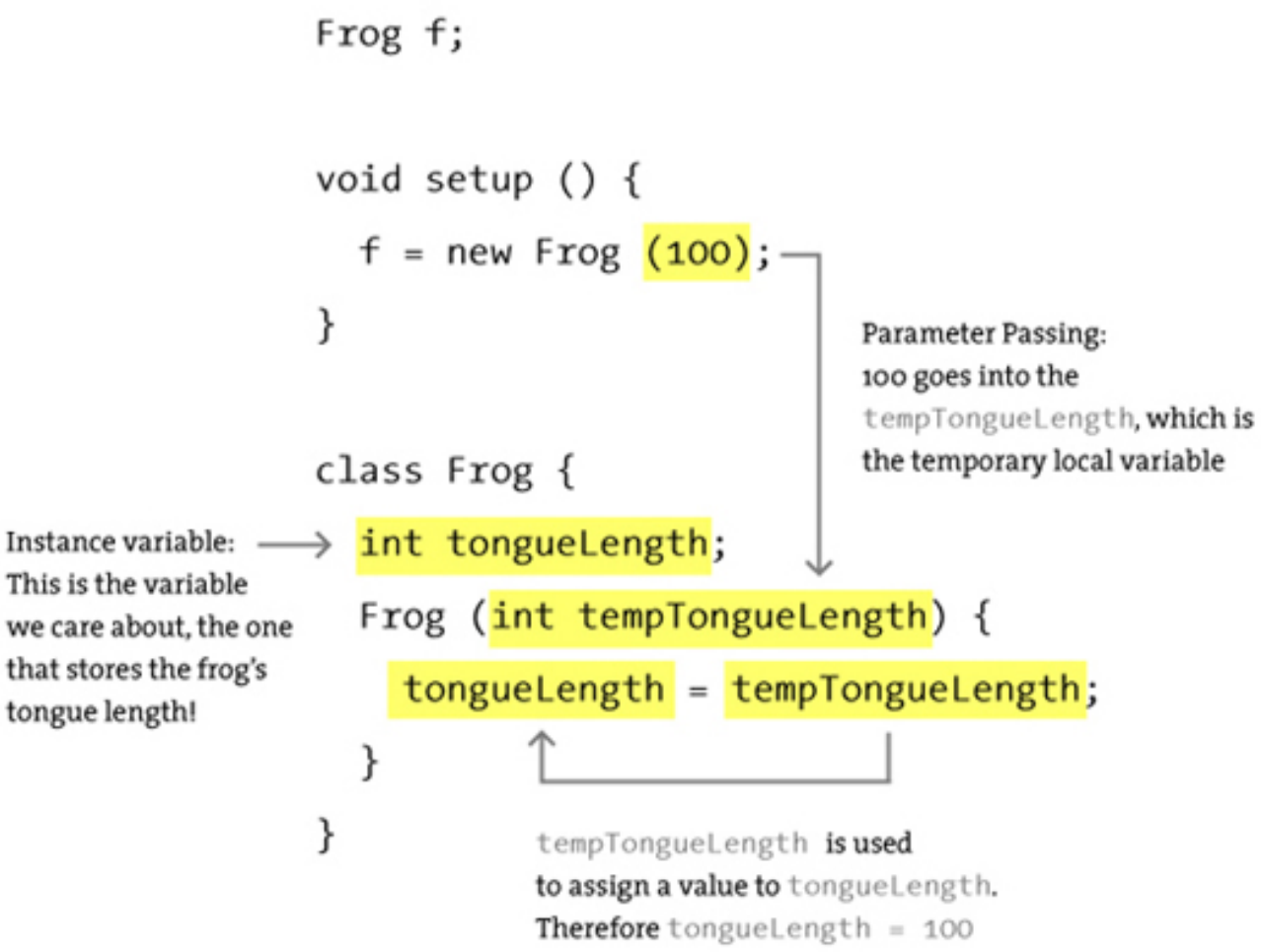

```
Car myCar1;
Car myCar2; // Two objects!

void setup() {
  size(200,200);
  // Parameters go inside the parentheses when the object is constructed.
  myCar1 = new Car(color(255,0,0),0,100);
  myCar2 = new Car(color(0,0,255),0,10);
}

void draw() {
  background(255);
  myCar1.drive();
  myCar1.display();
  myCar2.drive();
  myCar2.display();
}
```

Parameters in constructors

- To make the above code work we need our cars to look different
- Can add parameters to our constructors, used to initialize different cars differently

```
Car(color tempC, float tempXpos, float tempYpos) {
  c = tempC;
  xpos = tempXpos;
  ypos = tempYpos;
}
```

Constructor parameters can be confusing



Translation: Make a new frog with a tongue length of 100.

Scoping rules

- Remember "scoping" just means what variables are visible from where, in a program.
- Parameter variables names are only visible within the function that uses them
- Variables defined within a function are only visible within that function
- Object instance variables are visible anywhere in that instance (i.e., from any function in that object)
- Each object's instance variables are separate from each others
  - Even if you use the "cookie cutter" to stamp out multiple cookies, each one's data is separate
  - Even though the variable names are the same for all objects of the class.

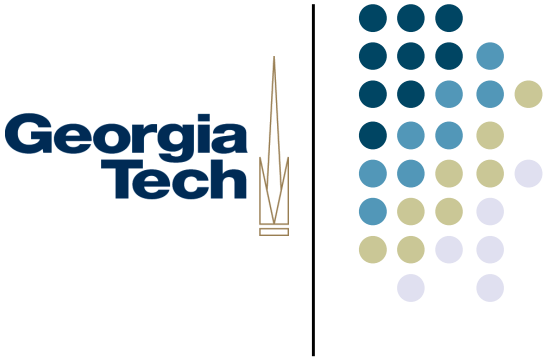
Objects are data

- Once you create a class, you're defining a new data type (just like integers, strings, etc.)
- Objects can contain other objects!
  - Just include it in the Class definition

```
class Garage {
  Car car1, car2;
}
```

- Objects can be passed as arguments to functions, just like other data.
- One difference:
  - When "simple" data types are passed in to a function, a copy is made of them. Changes to the data inside the function don't affect the copy outside.
  - With objects, a "reference" is passed in. Changes to the object inside the function **do** affect the original.

Programming Challenge



1. Update the Car class
   - Change the way the car is drawn (add some graphics for a passenger, for instance)
   - Add a **speed** variable, and use that to control the speed of the car when it drives
2. Use an **array** of cars to allow more cars in your program, without needing to keep separate variables for each