

15-104 Introduction to Computing for Creative Practice

Fall 2020

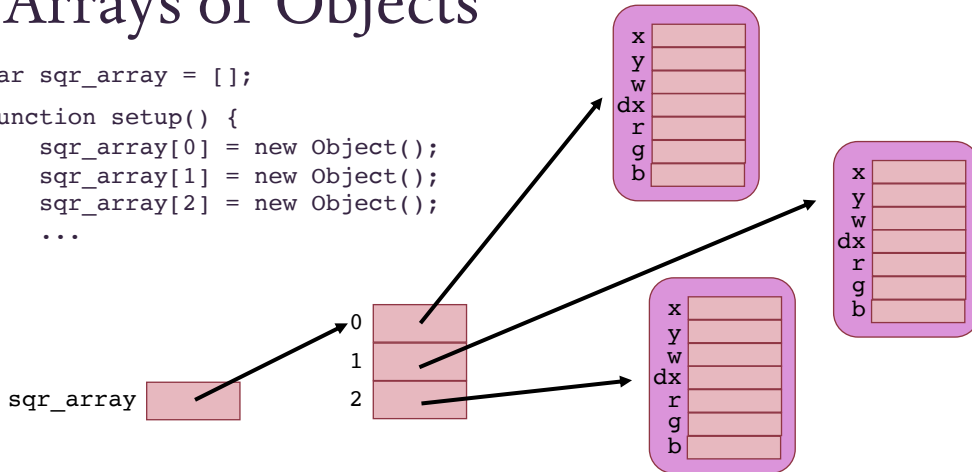
17 More Objects (based on examples from Khan Academy)

Instructor: Tom Cortina, tcortina@cs.cmu.edu, GHC 4117, 412-268-3514

1

Arrays of Objects

```
var sqr_array = [];  
function setup() {  
  sqr_array[0] = new Object();  
  sqr_array[1] = new Object();  
  sqr_array[2] = new Object();  
  ...  
}
```



2

Arrays of Objects

```

var sqr_array = [];
function setup() {
  sqr_array[0] = new Object();
  sqr_array[1] = new Object();
  sqr_array[2] = new Object();
  comp(sqr_array);
}
function comp(sa) {
  ...
}
sqr_array
sa

```

3

Functions as Values

- Recall we define functions in p5.js this way:


```

function area() {
  return width * height;
}

```
- We treat `area` as something special but it's just another variable.
- It's a global variable whose value is not a number or string or array, but instead its value is a function that can compute something.
- Since `area` is essentially a variable, we can assign another variable to it.


```

var canvasArea = area;
var x = canvasArea();

```

4



Anonymous Functions

- Some languages like Javascript allow for **anonymous functions**. These are functions that have no name, but they can be assigned to variables. Then when you use the variable as a function, you run the function.

```
var area = function() {
  return width * height;
}
function setup() {
  createCanvas(200, 300);
  print(area);
}
prints:
f() {
  return width * height;
}
```

```
var area = function() {
  return width * height;
}
function setup() {
  createCanvas(200, 300);
  print(area());
}
prints:
60000
```

5



Anonymous Functions

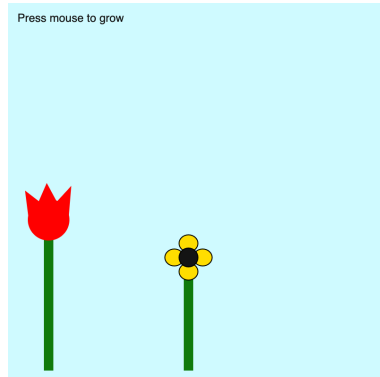
- Anonymous function can have parameters as well.

```
var volume = function(depth) {
  return width * height * depth;
}
function setup() {
  createCanvas(200, 300);
  print(volume);
}
prints:
f(depth) {
  return width * height * depth;
}
```

```
var volume = function(depth) {
  return width * height * depth;
}
function setup() {
  createCanvas(200, 300);
  print(volume(4));
}
prints:
240000
```

6

Example: Flower Grower (Khan Academy)



Properties of each flower:

- x
- y
- height

Behaviors of each flower:

- show itself (draw on the canvas)
- grow by some amount
- make itself (create or construct)

Objects not only can hold properties (fields) but they can also hold behaviors (functions) that they can perform on themselves.

7

this

- When we define a function that will be a behavior for an object, we will likely need to reference the properties of the object.
- We use the reference `this` to indicate that we are writing a function and we are referencing a property of this object while performing the function.
- Example: Suppose our object is a square box that has properties: `x`, `y`, and `w`, and we are programming a behavior `area` for the box that allows it to return its own area. We would write its function this way:

```
function area() {
  return this.w * this.w;
}
```

This function `area` would be stored as a property in the object alongside `x`, `y`, and `w`! Remember that functions are values too!

8



Tulip behaviors

```
function tulipDraw() {
  noStroke();
  fill(16, 122, 12);
  rect(this.x, this.y - this.height, 10, this.height);
  // petals
  fill(255, 0, 0); // red
  var y = this.y - this.height; // to save some code
  ellipse(this.x + 5, y, 44, 44);
  triangle(this.x - 16, y, this.x + 20, y, this.x - 20, y - 31);
  triangle(this.x - 14, y, this.x + 24, y, this.x + 3, y - 39);
  triangle(this.x + -4, y, this.x + 26, y, this.x + 29, y - 36);
};
```

9



Tulip behaviors (cont'd)

```
function tulipGrow(amount) {
  this.height += amount;
};

// makeTulip is a special behavior called a CONSTRUCTOR.
// It constructs an instance of the object.

function makeTulip(tx, ty, th) {
  var tulip = {x: tx, y: ty, height: th,
              show: tulipDraw, grow: tulipGrow}; // new object
  return tulip; // return the new object
};
```

10



Sunflower behaviors

```
function sunflowerDraw() {
  noStroke();
  fill(16, 122, 12);
  rect(this.x, this.y - this.height, 10, this.height);
  stroke(0, 0, 0);
  fill(255, 221, 0); // yellow
  var y = this.y - this.height;
  ellipse(this.x - 10, y, 20, 18);
  ellipse(this.x + 5, y - 15, 20, 18);
  ellipse(this.x + 5, y + 15, 20, 18);
  ellipse(this.x + 20, y, 20, 18);
  fill(20, 20, 20); // dark center
  ellipse(this.x + 5, y, 20, 20);
};
```

11



Sunflower behaviors (cont'd)

```
function sunflowerGrow(amount) {
  this.height += amount;
};

// makeSunflower is a special behavior called a CONSTRUCTOR.
// It constructs an instance of the object.

function makeSunflower(sx, sy, sh) {
  var sunflower = {x: sx, y: sy, height: sh,
                  show: sunflowerDraw, grow: sunflowerGrow};
  return sunflower; // return the new object
};
```

12

Main program

```

var tulip;
var sunflower;

function setup() {
  createCanvas(400, 400);
  tulip = makeTulip(38, 390, 150);
  sunflower = makeSunflower(186, 390, 100);
  frameRate(5);
}

```

```

function tulipDraw() {
  ...
}

function tulipGrow(amount) {
  ...
}

```

13

Main program (cont'd)

```

function draw() {
  background(207, 250, 255);
  tulip.show();
  sunflower.show();
  noStroke();
  text("Press mouse to grow", 10, 20);
};

function mousePressed() {
  tulip.grow(5);
  sunflower.grow(10);
}

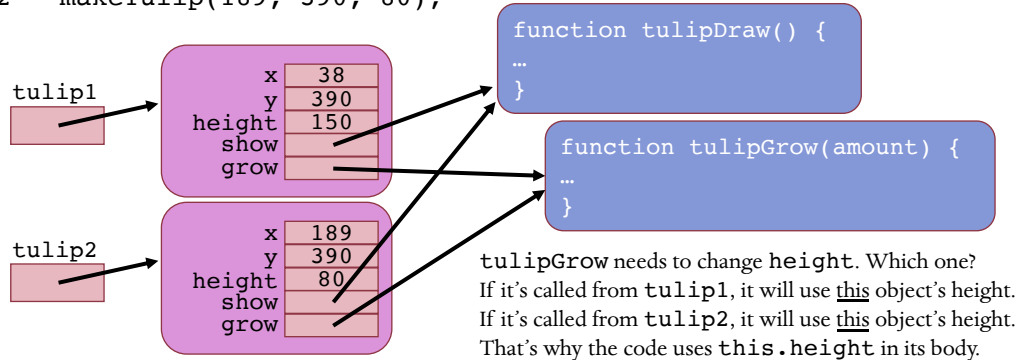
```

14

Revisiting `this`

Why do we need `this` in the methods?

```
tulip1 = makeTulip(38, 390, 150);
tulip2 = makeTulip(189, 390, 80);
```



15

Methods

The behaviors of an object are called **methods**.

Note that we defined tulip and sunflower to have the same method names: `show` and `grow`. (They call different functions, of course.)

This way, if we had a variable that was either a tulip or a sunflower, we could call `show` either way and it would work!

```
...
var flower = tulip;
flower.show();
flower = sunflower;
flower.show();
...
```

16

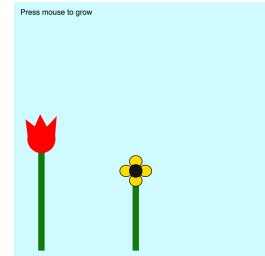
Array of 2 Flowers: Main program

```

var tulip;
var sunflower;
var flowers = []; // an empty array

function setup() {
  createCanvas(400, 400);
  tulip = makeTulip(38, 390, 150);
  sunflower = makeSunflower(186, 390, 100);
  flowers.push(tulip);
  flowers.push(sunflower);
  // or, you could write flowers = [tulip, sunflower];
  frameRate(5);
}

```



17

Array of 2 Flowers: Main program


```

function draw() {
  background(207, 250, 255);
  for (var i = 0; i < flowers.length; i++) {
    flowers[i].show();
  }
  noStroke();
  text("Press mouse to grow", 10, 20);
};

function mousePressed() {
  for (var i = 0; i < flowers.length; i++) {
    flowers[i].grow(5);
  }
}

```

18




Random Acts of Flowers (Main Program)

```


var tulip;
var sunflower;
var flowers = []; // an empty array

function setup() {
  createCanvas(400, 400);
  tulip = makeTulip(38, 390, 150);
  sunflower = makeSunflower(186, 390, 100);
  flowers.push(tulip);
  flowers.push(sunflower);
  frameRate(5);
}

```



19



Random Acts of Flowers (Main Program)

```

function draw() {
  background(207, 250, 255);
  for (var i = 0; i < flowers.length; i++) {
    flowers[i].show();
  }
  noStroke();
  fill(0);
  text("Press key for acts of random flowers.", 10, 20);
  text("Press mouse to grow", 10, 35);
};

```

20

Random Acts of Flowers (Main Program)

```
function mousePressed() {
  for (var i = 0; i < flowers.length; i++) {
    flowers[i].grow(5);
  }
}

function keyPressed() {
  // allFlowers is an array of constructors!
  var allFlowers = [makeTulip, makeSunflower];
  // flowerMaker is a random constructor from the array
  var flowerMaker = random(allFlowers);
  flowers.push(flowerMaker(random(width),
    height - random(100),
    90 + random(50)));
}
```

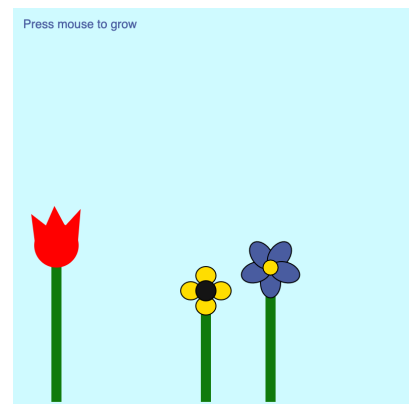
21

Try This

Add a violet to the previous programs.

Give each flower a different amount to grow by when the mouse is clicked.

Modify the random flower generator so it displays only the most recent 10 flowers.



22