imc

js p5*

introduction to media computing
week 06

School of creative media, mw

Today’s topics (week 06)

js

- quick review
- Arrays Part 1 - introduction to arrays

Today’s topics (week 06)

js

- quick review
- Arrays Part 1 - introduction to arrays

p5*

- utility function `lerp()`

Review: `for()` loop

A concise construct for looping

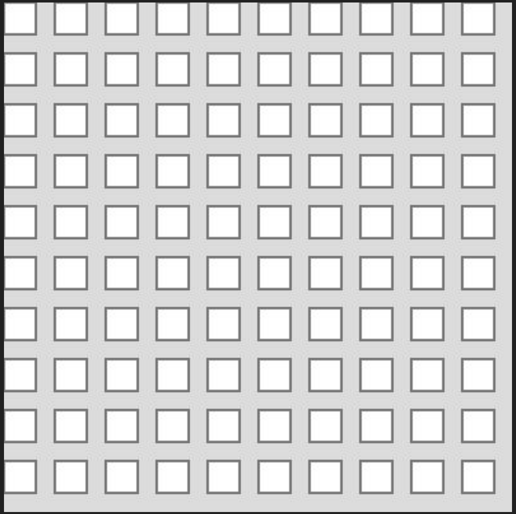
```
for (let i = 0; i < 10; i++) {
  // Some Code here
}
```

```
for (<init>; <cond>; <iter>) {
  // Some Code here
}
```

Review: Nested `for()` loop

Nested use of `for()` loop

```
for (let y = 0; y < 10; y++) {
  for (let x = 0; x < 10; x++) {
    rect(x*40, y *40, 25, 25);
  }
}
```



Review: simple logging `console.log()`

`console.log()` helps us to keep track of variables

Example:

```
   console.log(x);
```

prints the value of variable `'x'` to the console.

** where is the console ? **

```
console.log(x);
```

Review: more mouse `pmouseX,pmouseY`

`pmouseX` and `pmouseY` store the values of `mouseX` and `mouseY` of previous frame (drawn by `function draw()`) respectively.

Example: to measure the mouse displacement:

```
dist(pmouseX, pmouseY, mouseX, mouseY);
```

Review: measure time with `millis()`

`millis()` returns the number of milliseconds since the program has started.

(1 millisecond = 0.001 sec)

[ ] Array Part 1
Introduction to Arrays

Array: introduction

In our daily life, we often group similar data into an indexed list for convenience such as a student list.

Student No.	**Name**
0	Annie Lennox
1	Matt Pharr
2	Paul Smith
...	...

Array: introduction

In our daily life, we often group similar data into an indexed list for convenience such as a student list.

Why?

Student No.	**Name**
0	Annie Lennox
1	Matt Pharr
2	Paul Smith
...	...

Array: introduction

In our daily life, we often group similar data into an indexed list for convenience such as a student list.

Why?

Student No.	Name
0	Annie Lennox
1	Matt Pharr
2	Paul Smith
...	...

Ease of reference and organization.

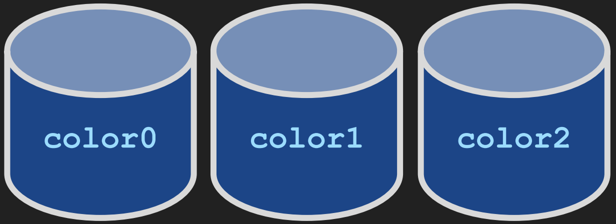
Array: introduction

A simple variable is a *named* container of a single piece of data.

```
let activeColor = 0;
let color0 = 255;

if (activeColor == 0) {
  fill(color0);
}
```

Array: introduction



Getting repetitive and triesome …..

```
let activeColor = 0;
let color0 = 255;
let color1 = 200;
let color2 = 150;

if (activeColor == 0) {
  fill(color0);
}
else if (activeColor == 1) {
  fill(color1);
}
else if (activeColor == 2) {
  fill(color2);
}
```

Array: introduction

js

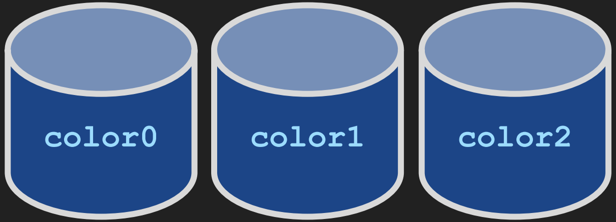
3 Simple variables

color0
color1
color2

```
let color0 = 255;
let color1 = 200;
let color2 = 150;
```

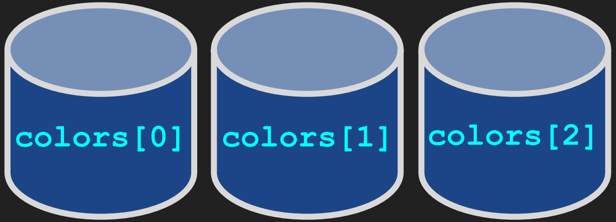
Array: An indexed collection of variables

3 Simple variables



```
let color0 = 255;
let color1 = 200;
let color2 = 150;
```

An ARRAY with 3 members



Each member is accessed by its index

```
let colors = [];
colors[0] = 255;
colors[1] = 200;
colors[2] = 150;
```

Array: An indexed collection of variables

```
let activeColor = 0;
let color0 = 255;
let color1 = 200;
let color2 = 150;

if (activeColor == 0) {
  fill(color0);
}
else if (activeColor == 1) {
  fill(color1);
}
else if (activeColor == 2) {
  fill(color2);
}
```

```
let activeColor = 0;
let colors = [];
colors[0] = 255;
colors[1] = 200;
colors[2] = 150;

fill( colors[activeColor] );
```

Array: An indexed collection of variables

```
let activeColor = 0;
let color0 = 255;
let color1 = 200;
let color2 = 150;

if (activeColor == 0) {
  fill(color0);
}
else if (activeColor == 1) {
  fill(color1);
}
else if (activeColor == 2) {
  fill(color2);
}
```

```
let activeColor = 0;
let colors = [];
colors[0] = 255;
colors[1] = 200;
colors[2] = 150;

fill( colors[activeColor] );
```

When `activeColor` equals to 1,

then we are filling with `colors[1]`.

Array: more examples

An array of strings for a name list.

Student No.	Name
0	Annie Lennox
1	Matt Pharr
2	Paul Smith
...	...

```
let names = [];
names[0] = "Annie Lennox";
names[1] = "Matt Pharr";
names[2] = "Paul Smith";
```

Array: more examples

To keep track of the positions of objects, i.e. one array for their X-coordinates, and one array for their Y-coordinates

rect #	X-coord	Y-coord
0	10	20
1	100	110
2	200	210
...	...	

```
let posX = [];
let posY = [];
posX[0] = 10;
posY[0] = 20;
posX[1] = 100;
posY[1] = 110;
posX[2] = 200;
posY[2] = 210;
```

Array: more examples

To keep track of the positions of objects,
i.e. one array for their X-coordinates,
and one array for their Y-coordinates

rect #	X-coord	Y-coord
0	10	20
1	100	110
2	200	210
...	...	

```
let posX = [];
let posY = [];
posX[0] = 10;
posY[0] = 20;
posX[1] = 100;
posY[1] = 110;
posX[2] = 200;
posY[2] = 210;
```

```
for (let i=0; i<3; i++) {
  rect(posX[i],posY[i],10,10);
}
```

Array and loop go together !

Array: Declaration & Initialization

js

To declare an array:

```
let posX = [];
```

To declare an array and initialize it at the same time:

```
let posX = [10,20,30];
```

Array: Declaration & Initialization

To declare an array:

```
let posX = [];
```

To declare an array and initialize it at the same time:

```
let posX = [10,20,30];
```

```
let posX = [];
posX[0] = 10;
posX[1] = 20;
posX[2] = 30;
```

Array: Access

To access a member of an array, supply an index:

via a constant

```
fill(colors[2]);
```

Array: Access

To access a member of an array, supply an index:

via a constant

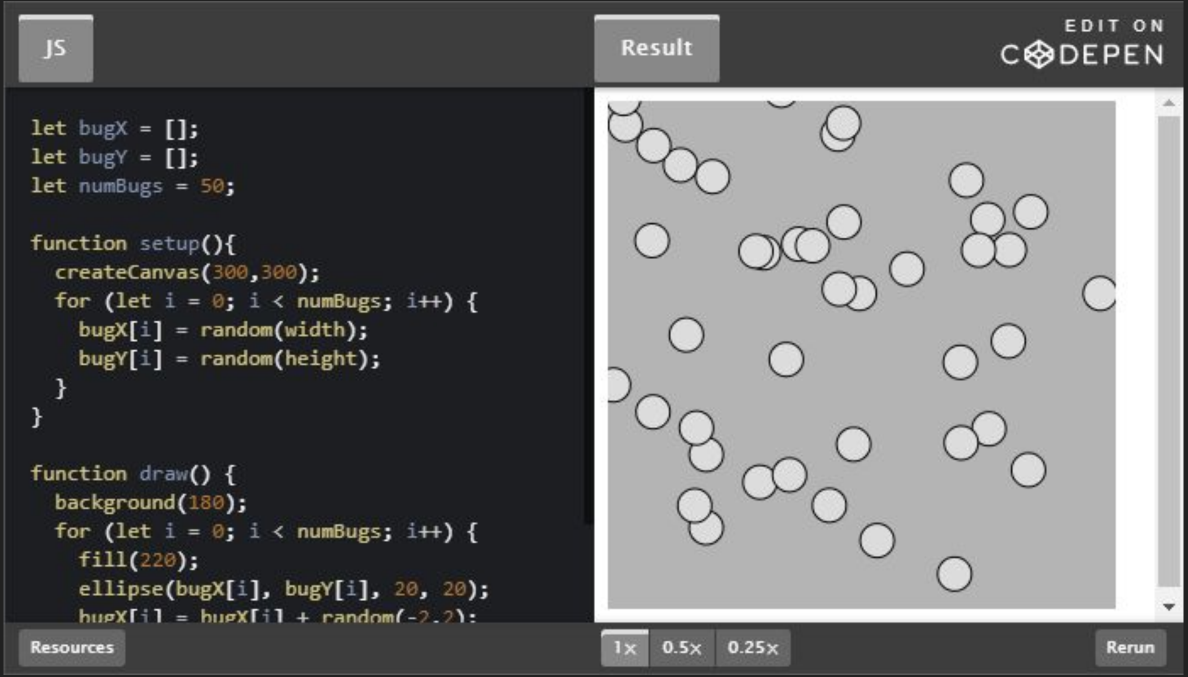
```
fill(colors[2]);
```

via a variable

```
fill(colors[i]);
```

Simple particle system using arrays

Use multiple arrays to store the (x,y) coordinate of each particle.



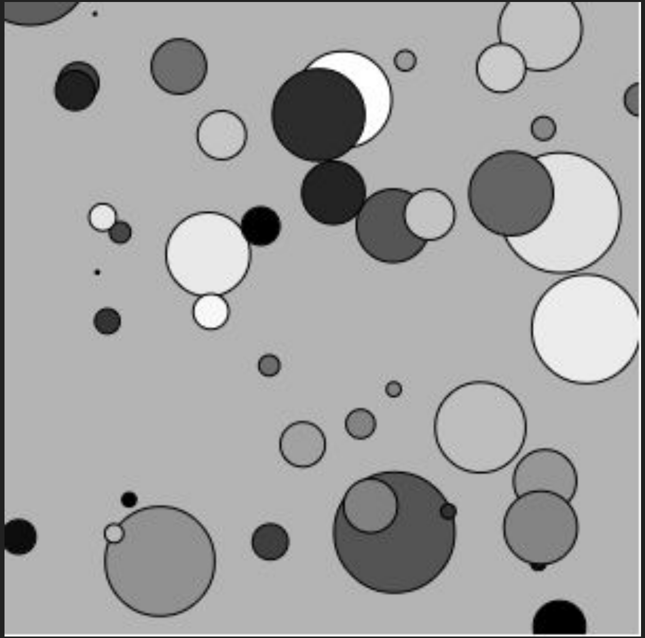
```
let bugX = [];
let bugY = [];
let numBugs = 50;

function setup(){
  createCanvas(300,300);
  for (let i = 0; i < numBugs; i++) {
    bugX[i] = random(width);
    bugY[i] = random(height);
  }
}

function draw() {
  background(180);
  for (let i = 0; i < numBugs; i++) {
    fill(220);
    ellipse(bugX[i], bugY[i], 20, 20);
    bugX[i] = bugX[i] + random(-2,2);
```

exercise

In-class exercise 1



Based on the simple particle system example (on Canvas page), add additional arrays such that the particles also vary in both size and color in each iteration.

Utility function `lerp()`

`lerp(start, stop, amount)`

`lerp()` computes a number between the `'start'` and `'end'` values determined by the given `'amount'` (a value between 0.0 to 1.0).

Examples:

`lerp(0, 10, 0.0)` returns `0`

`lerp(0, 10, 1.0)` returns `10`

`lerp(0, 10, 0.25)` returns `2.5`

`lerp(0, 10, 0.5)` returns `5`

`lerp(start, stop, amount)`

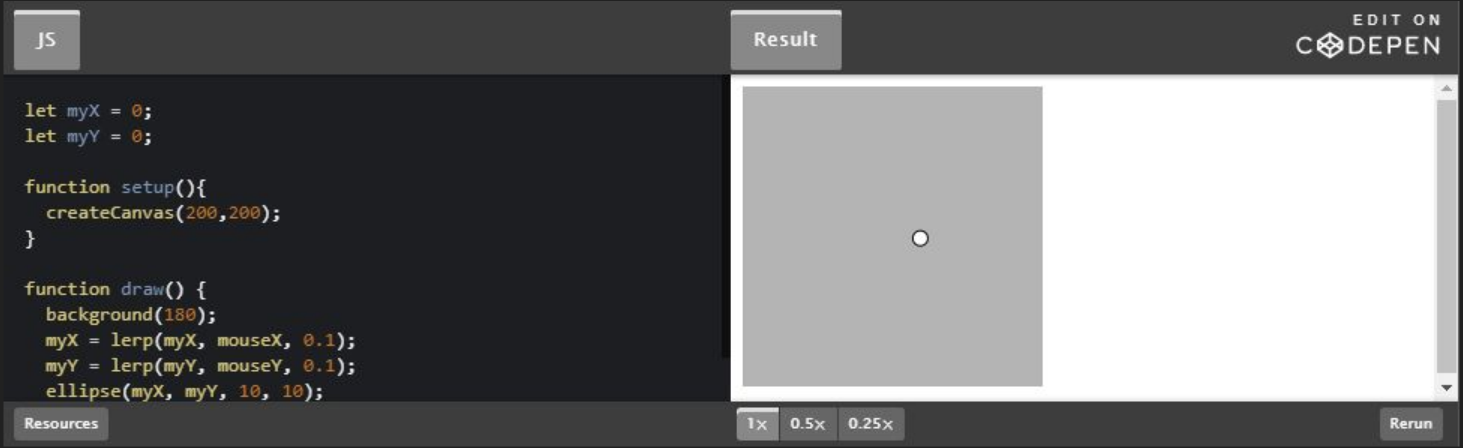
Examples use:

`lerp(mouseX, pmouseX, 0.5);` returns a value which is in the middle between `mouseX` and `pmouseX.` So we may use the following to compute a (x,y) coordinate between previous and present mouse cursor positions.

```
myX = lerp(mouseX, pmouseX, 0.5);
myY = lerp(mouseY, pmouseY, 0.5);
```

Natural Motion using `lerp()`

Use `lerp()` to create an impression of delayed mouse tracking.



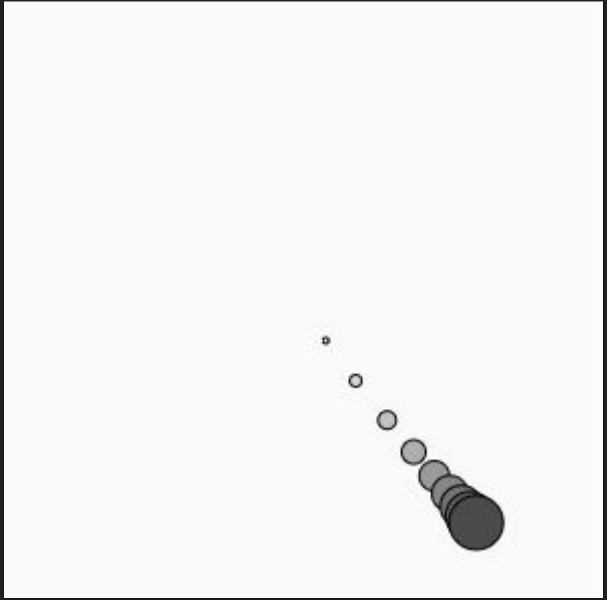
```
let myX = 0;
let myY = 0;

function setup(){
  createCanvas(200,200);
}

function draw() {
  background(180);
  myX = lerp(myX, mouseX, 0.1);
  myY = lerp(myY, mouseY, 0.1);
  ellipse(myX, myY, 10, 10);
```

exercise

In-class exercise 2



Use the previous 'Natural Motion' example as your skeleton code; and add additional particles (using array).

For each particle, apply different `'amount'` in the `lerp()` function to create a trail of particles similar to the sample shown on the canvas page.