imc

js p5*

introduction to media computing
week 10

School of creative media, mw

Today’s topics (week 10)

- Function Part 1

Today’s topics (week 10)

js

- Function Part 1

p5*

- Quick review of font and image resources
- Data resources

Function Part 1

Function: Introduction

We have been using various p5.js built-in 'functions' in our sketches, they include:

```
noFill();
noStroke();
...
```

```
point(10,10);
stroke(255);
...
```

```
let y = floor(x);
let p = lerp(a,b,0.1);
...
```

Function: Introduction

We have been using various p5.js built-in 'functions' in our sketches, they include:

```
noFill();
noStroke();
...
```

```
point(10,10);
stroke(255);
...
```

```
let y = floor(x);
let p = lerp(a,b,0.1);
...
```

Function: Introduction

We have been using various p5.js built-in 'functions' in our sketches, they include:

```
noFill();
noStroke();
...
```

```
point(10,10);
stroke(255);
...
```

```
let y = floor(x);
let p = lerp(a,b,0.1);
...
```

1. Simple functions

Function: Introduction

We have been using various p5.js built-in 'functions' in our sketches, they include:

```
noFill();
noStroke();
...
```

```
point(10,10);
stroke(255);
...
```

```
let y = floor(x);
let p = lerp(a,b,0.1);
...
```

1. Simple functions

2. Functions which take parameters

Function: Introduction

We have been using various p5.js built-in 'functions' in our sketches, they include:

```
noFill();
noStroke();
...
```

1. Simple functions

```
point(10,10);
stroke(255);
...
```

2. Functions which take parameters

```
let y = floor(x);
let p = lerp(a,b,0.1);
...
```

3. Functions which take parameters, and return values

js

Function: Write our own simple function

1. function can be regarded as a Named Block of Code.
2. serves a dedicated purpose.
3. replaces repetitive code.

Function: Write our own simple function

1. function can be regarded as a Named Block of Code.
2. serves a dedicated purpose.
3. replaces repetitive code.

```
stroke(0);
strokeWeight(5);
noFill();
rect(10,10,100,100);
```

Function: Write our own simple function

1. function can be regarded as a Named Block of Code.
2. serves a dedicated purpose.
3. replaces repetitive code.

1

drawing instructions
that we use repeatedly.

```
stroke(0);
strokeWeight(5);
noFill();
rect(10,10,100,100);
```

Function: Write our own simple function

1. function can be regarded as a Named Block of Code.
2. serves a dedicated purpose.
3. replaces repetitive code.

1 drawing instructions that we use repeatedly.

```
stroke(0);
strokeWeight(5);
noFill();
rect(10,10,100,100);
```

2 Make that block as a simple function named `myStyle()`.

```
function myStyle() {
  stroke(0);
  strokeWeight(5);
  noFill();
}
```

Function: Write our own simple function

1. function can be regarded as a Named Block of Code.

2. serves a dedicated purpose.

3. replaces repetitive code.

1 drawing instructions that we use repeatedly.

```
stroke(0);
strokeWeight(5);
noFill();
rect(10,10,100,100);
```

2 Make that block as a simple function named `myStyle()`.

```
function myStyle() {
  stroke(0);
  strokeWeight(5);
  noFill();
}
```

3 Substitute the block by our new `myStyle()`.

```
myStyle();
rect(10,10,100,100);
```

Function: Write our own simple function

js

1 drawing instructions that we use repeatedly.

```
stroke(0);
strokeWeight(5);
noFill();
rect(10,10,100,100);
```

2 Make that block as a simple function named `myStyle()`.

```
function myStyle() {
  stroke(0);
  strokeWeight(5);
  noFill();
}
```

3 Substitute the block by our new `myStyle()`.

```
myStyle();
rect(10,10,100,100);
```

Function: Write our own simple function

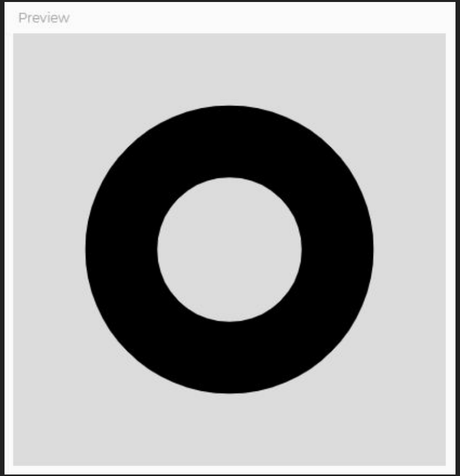
```
function <name>() {
   // code here
}
```

Function: function with parameters

```
function <name>(param1,param2,..){
  // code here
}
```

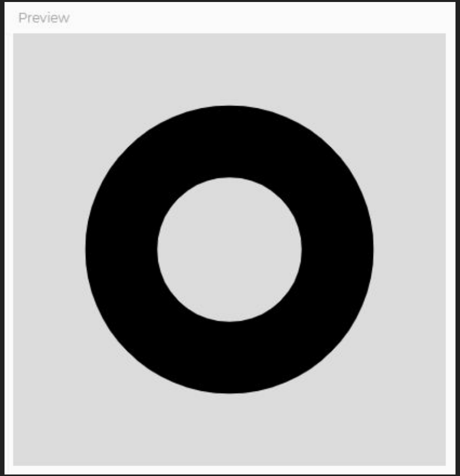
Function: function with parameters

```
function donut(x,y,size) {
  noFill();
  stroke(0);
  strokeWeight(size/3);
  ellipse(x,y,size);
}

function setup() {
  createCanvas(400, 400);
  background(220);
  donut(200,200,200);
}
```



Function: function with parameters

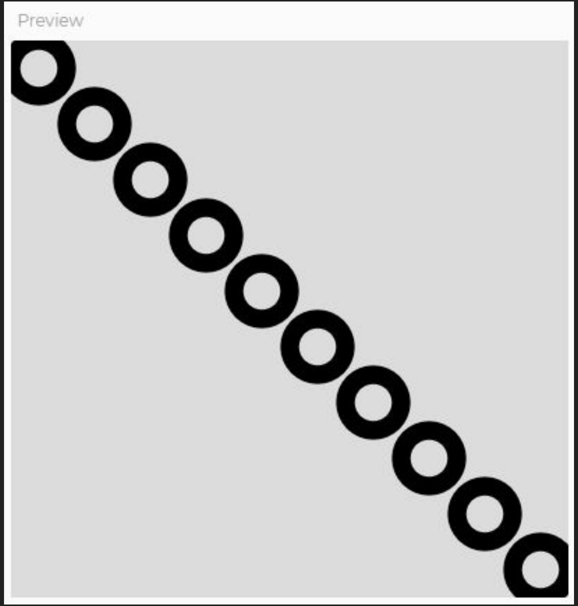
```
function donut(x,y,size) {
  noFill();
  stroke(0);
  strokeWeight(size/3);
  ellipse(x,y,size);
}

function setup() {
  createCanvas(400, 400);
  background(220);
  donut(200,200,200);
}
```



we call `donut()` by passing constant values

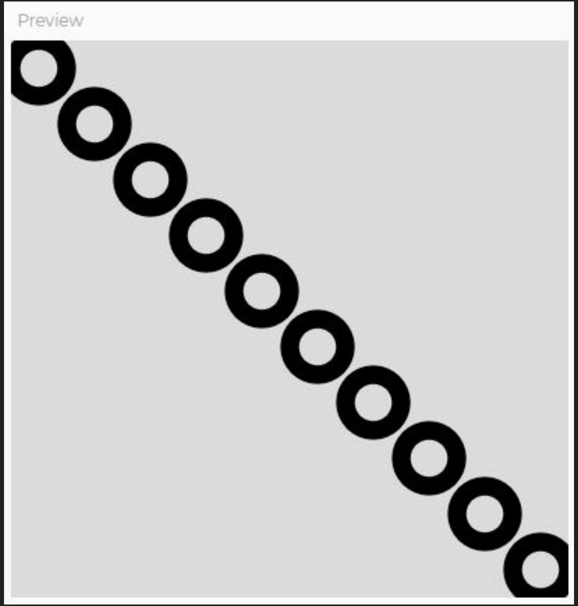
Function: function with parameters

```
function donut(x,y,size) {
  noFill();
  stroke(0);
  strokeWeight(size/3);
  ellipse(x,y,size);
}

function setup() {
  createCanvas(400, 400);
  background(220);
  for (let i = 20; i < 400; i+=40) {
    donut(i,i,40);
  }
}
```



Function: function with parameters

```
function donut(x,y,size) {
  noFill();
  stroke(0);
  strokeWeight(size/3);
  ellipse(x,y,size);
}

function setup() {
  createCanvas(400, 400);
  background(220);
  for (let i = 20; i < 400; i+=40) {
    donut(i,i,40);
  }
}
```

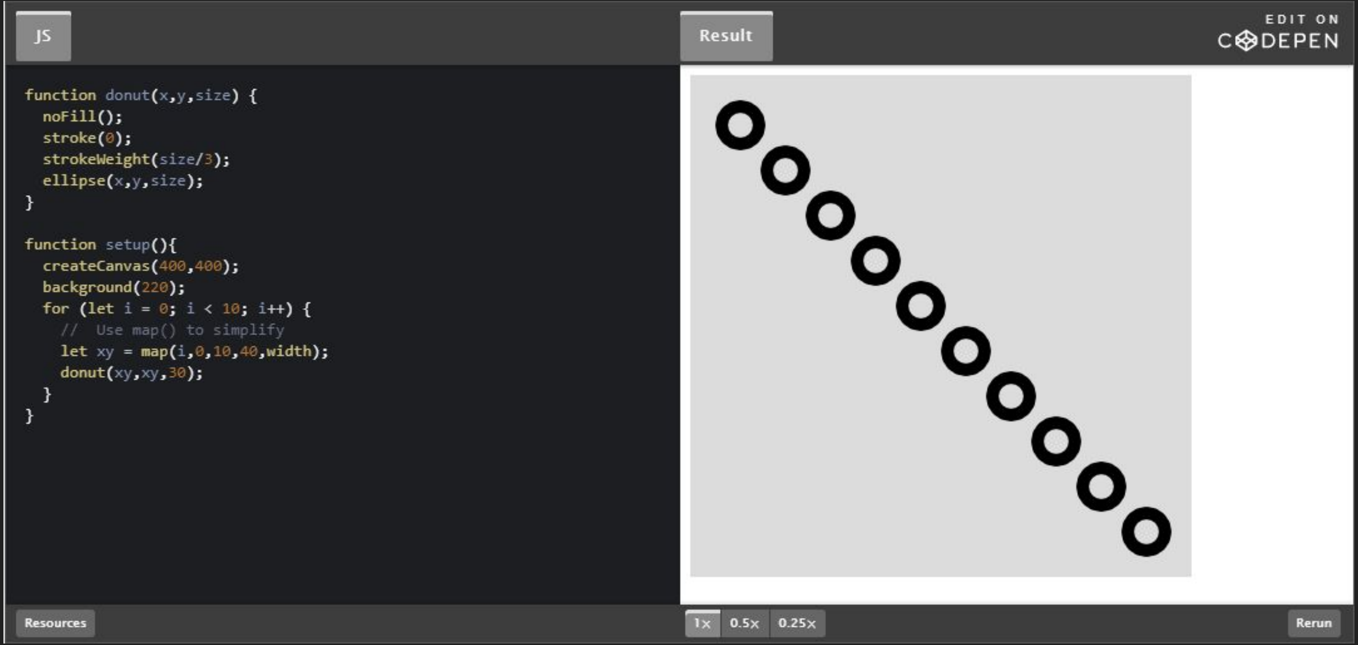


we may also call donut() by passing variables & constant

Example: function with parameters

try p5*

Example: function with parameters and the use of `map()` to relate loop variable to canvas coordinates.

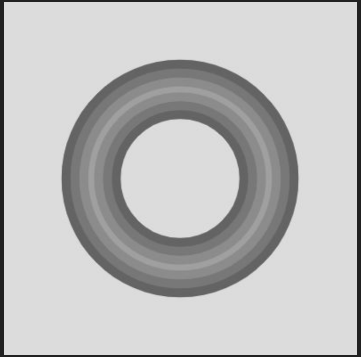


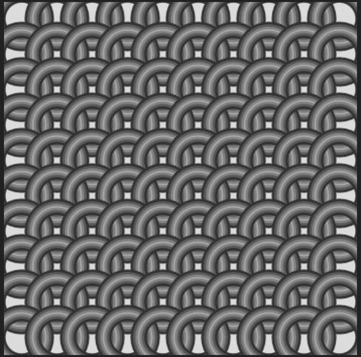
```
function donut(x,y,size) {
  noFill();
  stroke(0);
  strokeWeight(size/3);
  ellipse(x,y,size);
}

function setup(){
  createCanvas(400,400);
  background(220);
  for (let i = 0; i < 10; i++) {
    //  Use map() to simplify
    let xy = map(i,0,10,40,width);
    donut(xy,xy,30);
  }
}
```

CityU canvas

exercise

In-class exercise 1





1. Assume a canvas of size 400 x 400. Write your own `donut()` function such that it accepts 4 parameters as follows:

```
function donut(x,y,size,ring)
```

where `ring` is the number of rings on the donut shape (see figure). (Hint: decrease the parameter value to `strokeWeight()` by `'size/10'` in each step).

2. Use a double for-loop to draw a grid of donut (see figure) using your own `donut()` function.

Quick Review

Review: round(), ceil() & floor()

When we work with 'array', we have to access its members via an index which is an integer. It is quite often that we want to get an integer from a given decimal number.

function	returns	examples

`round(x)`	a rounded number	`round(3.4)-> 3`
		`round(3.5)-> 4`

`floor(x)`	the largest integer < `x`
	i.e. floor of `x`
		`floor(3.4)-> 3`
		`floor(3.5)-> 3`

`ceil(x)`	the smallest integer > `x`
	i.e. ceiling of `x`
		`ceil(3.4)-> 4`
		`ceil(3.5)-> 4`

Review: `preload()`

```
<html>
<head>
  <script src="https://cdnjs.cloudflare.com/ajax/libs/p5.js/0.9.0/p5.js"></script>
  <script>
  function preload() {
    font01 = loadFont('font01.ttf');
  }
  function setup() {
  }
  function draw() {
  }
  </script>
</head>
</html>
```

Review: `loadFont()`

p5.js supports OpenType (.otf) and TrueType (.ttf) fonts. Fonts must be loaded via using `loadFont()` inside the function `preload()` before using them. The font file may be a local file or a URL served by a web server.

```
let font01, font02;
function preload() {
  font01 = loadFont('./fontNumberOne.ttf');
  font02 = loadFont('https://someserver.com/somefont.ttf');
}
```

Review: `textFont()`

Once a font has been loaded. You may use the `textFont()` function to define the desired font to be used with the `text()` function.

```
function draw() {
  textFont(font02);
  text( "Hello", 30, 30);
  ...
}
```

Review: `loadImage()`

p5.js supports major image types. Images should be loaded via using `loadImage()` inside the function `preload()` before using them. The image file may be a <u>local file</u> or a <u>URL served by a web server</u>.

```
let image01, image02;
function preload() {
  image01 = loadImage('./someImage.jpg');
  image02 = loadImage('https://someserver.com/someimage.jpg');
}
```

Review: image()

Once the images are pre-loaded using loadImage(). We may use image() to display them. image() supports:

```
image(img,x,y);//(x,y) upper left coord.
image(img,x,y,[width],[height]);
// width,height: scaled width & height on canvas.
```

```
function draw() {
  image( image01, 100, 100 );
  image( image01, 0, 0, 100, 100 );
}
```

p5*

Data Resource

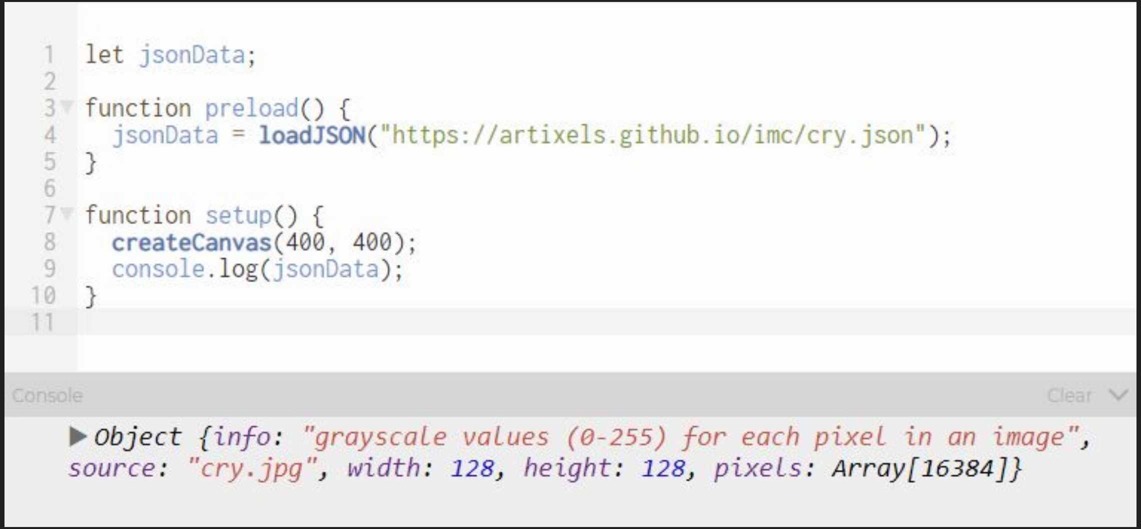
Data Source: Data in JSON format

Apart from image and font, p5.js can also import generic data via the JSON (JavaScript Object Notation) format. All data resource must be loaded in `preload()`.

```
let jsonData1, jsonData2;
function preload() {
  jsonData1 = loadJSON('./data1.json');
  jsonData2 = loadJSON('https://someserver.com/data2.json');
}
```

Data Source: Data in JSON format

An example of loading a picture stored in JSON format. We may use `console.log()` to inspect its content.



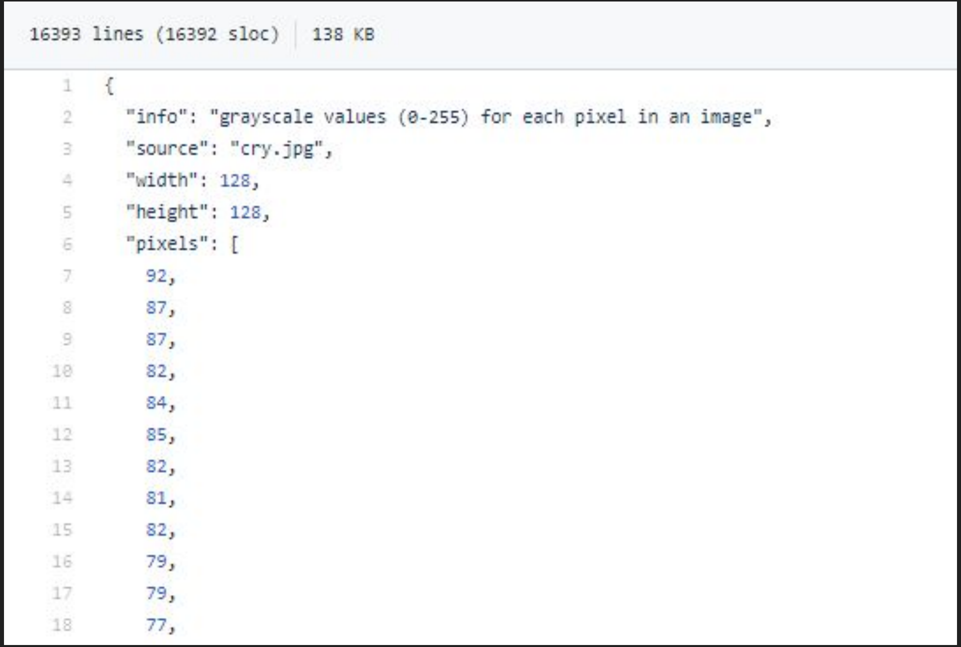
```
let jsonData;

function preload() {
  jsonData = loadJSON("https://artixels.github.io/imc/cry.json");
}

function setup() {
  createCanvas(400, 400);
  console.log(jsonData);
}

```

```
▶Object {info: "grayscale values (0-255) for each pixel in an image",
source: "cry.jpg", width: 128, height: 128, pixels: Array[16384]}
```

attributes
`.info`
`.source`
`.width`
`.height`
`.pixels`

Data Source: JSON file viewed as text

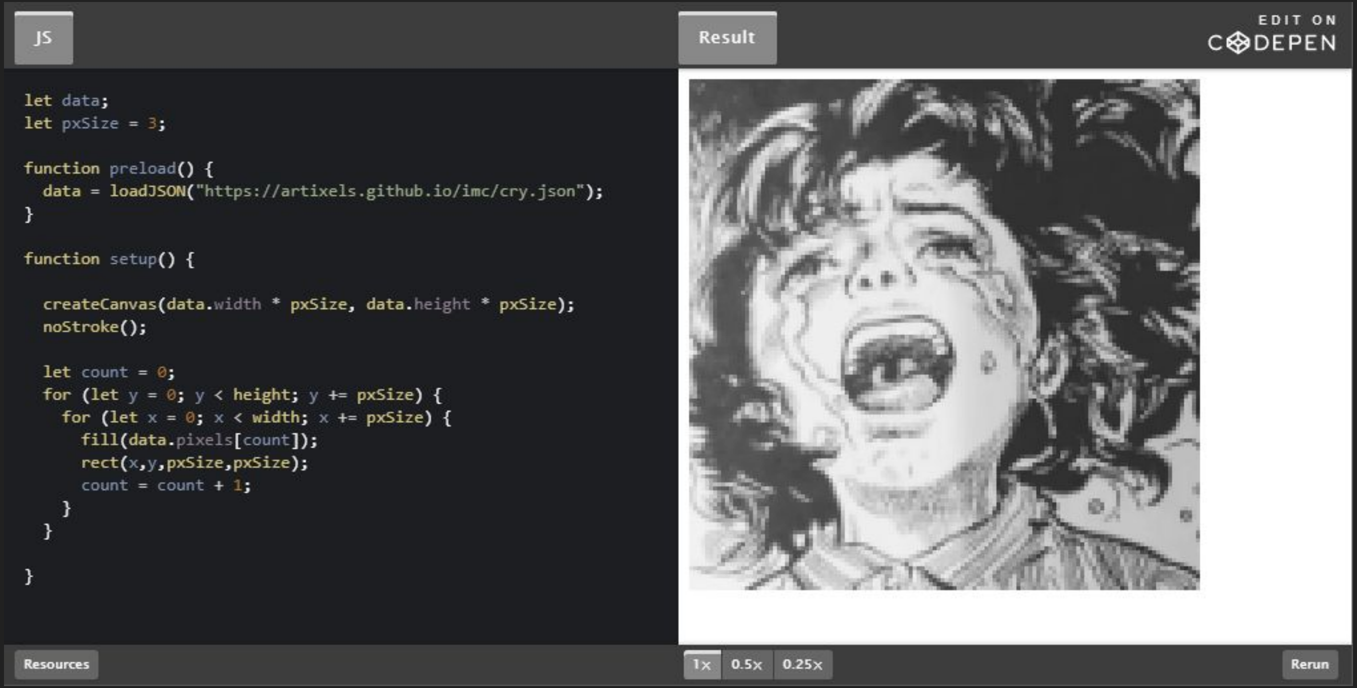
Most JSON data file can be inspected in text editors too.



```
16393 lines (16392 sloc) | 138 KB
```

```
{
  "info": "grayscale values (0-255) for each pixel in an image",
  "source": "cry.jpg",
  "width": 128,
  "height": 128,
  "pixels": [
    92,
    87,
    87,
    82,
    84,
    85,
    82,
    81,
    82,
    79,
    79,
    77,
```

Data Source: JSON data visualized



```
let data;
let pxSize = 3;

function preload() {
  data = loadJSON("https://artixels.github.io/imc/cry.json");
}

function setup() {

  createCanvas(data.width * pxSize, data.height * pxSize);
  noStroke();

  let count = 0;
  for (let y = 0; y < height; y += pxSize) {
    for (let x = 0; x < width; x += pxSize) {
      fill(data.pixels[count]);
      rect(x,y,pxSize,pxSize);
      count = count + 1;
    }
  }

}
```

exercise

In-class exercise 2

js p5*

Use the previous JSON loading example as your skeleton, and use the following source as your JSON data:

`https://artixels.github.io/imc/poke.json`

Use `console.log()` function to inspect carefully what attributes and data it has, and write a sketch to display the stored picture at your own style such as reuse your `donut()` function.

CityU canvas