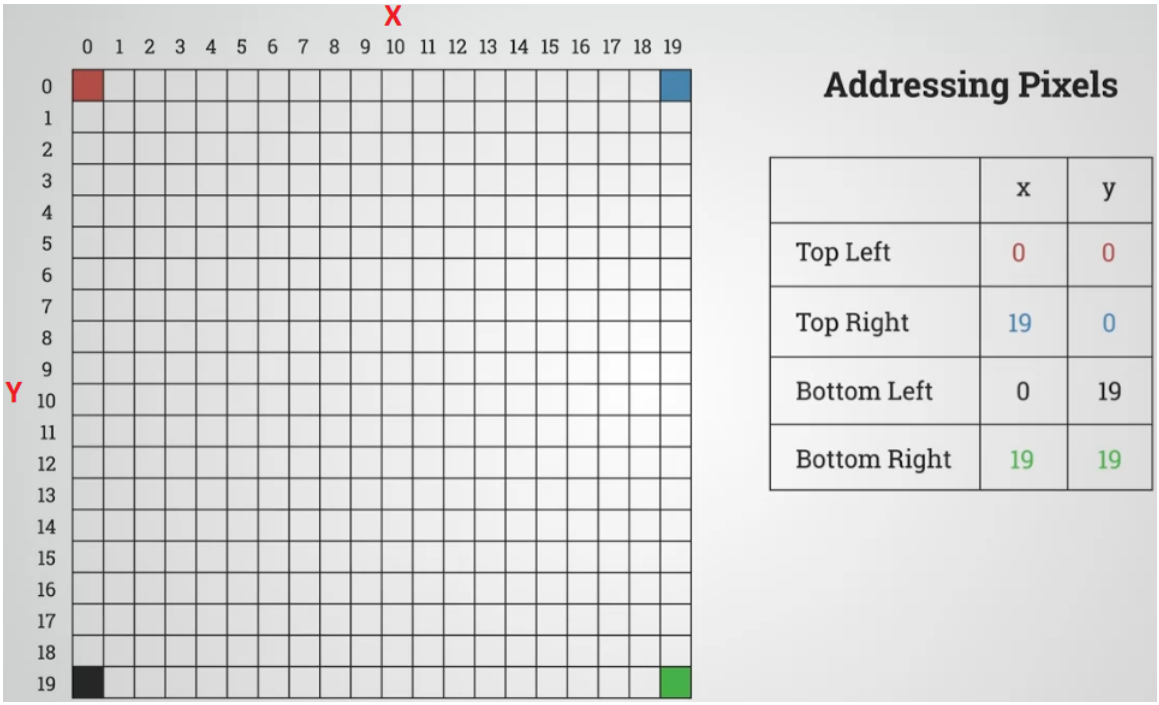
Intro to Prog I

p5.js

`preload()` and `setup()` run once, `draw()` repeats every frame



create area: `createCanvas(width,height);`

`background(r,g,b);`

`fill(r, g, b, alpha)` for inside color, `noFill();` disable inside colors

`stroke(r,g,b)` for edges, `noStroke();` disable edge colors, `strokeWeight(n);` for thicker edges

`rect(x,y,width,height);` where `x,y` is the top left corner (additional arg for roundness)

`ellipse(x,y,width,height);` where `x,y` is center (circle for width=height)

`triangle(x1,y1, x2,y2, x3,y3);`

`line(x1,y1, x2,y2);`

`point(x1,y1);` can be made more visible with `strokeWeight(n);` where default `n` = 1 (don't forget to reset afterwards)

to draw custom shapes:

`beginShape(kind);` see ref for kind options

`vertex(x, y);`

...

`vertex(x,y);`

`endShape(CLOSE);` remove `CLOSE` for not connect start and finish points

text

`text("message",x,y)`, `fill()` determines color

`textSize(n)` for font size

`textAlign(CENTER, CENTER);`

`textFont(FontName);` where `FontName = loadFont('font_file.otf');` in `preload()`

`textWidth("xyz")` returns pixel width of string

`img = loadImage('path/to/image.jpg');` load in preload()

`image(img, x, y);` display (optional with width and height)

`constrain(input, low_limit, high_limit)` returns only between limits

`map(input, current_low_limit, current_high_limit, wanted_low_limit, wanted_high_limit)` scales range

Variables

declare `var varName;` (value: undefined) and initialize it with `varName = ...;` in setup

`typeof()` to check type

Numbers

`>`, `>=`, `<`, `<=`,

`parseInt(num1)` returns integer

`random(min,max);` `min(num1,num2);` `max(num1,num2);`

Objects

declare: `var objectName`; at the beginning and initialize with `objectName = {propertyName_1: ..., propertyName_2: ..., anotherProperty_3: ...}` in setup and use `objectName.propertyName_2`

methods: `var objectName = {prop1: ..., methodName: function(input){} }` called by `objectName.methodName(x);`, use object's properties inside the method by `this.x`

`obj2 = obj1` just copies the reference, to truly copy `obj2 = {x: obj1.x, y: obj1.y ...}`

Arrays

declare: `var myArray = [];` or `[1,2,3,4,5]` or `[1, 2.32, "hello", false]` or `[[1,2],[3,4]]`

access: `myArray[5] = 10;`, `myArray[2][1]`, myArray `myArray.length`,

`myArray.push(value)` appends value to the end

`myArray.splice(i, 1);` removes element at index i

`for(var i=myArray.length; i≤0; i--){myArray.splice(i,1)}` to iterate and remove

Other Types

boolean `var varName =` can be `true` or `false`, `!varName` returns opposite

string can be enclosed in `""` or `''`, concat: `string1 + string2`

built-in variables

`width` & `height` canvas size

`mouseX` & `mouseY` cursor location

`key` pressed key (always in capital letters) or `keyCode` number

`frameCount` frame count

```
if (condition) {action}
else if (condition) {action}
else {action}
```

conditions: `==` is, `!=` is not, `&&` and, `||` or

ex: `dist(mouseX, mouseY, point.x, point.y) <= 15`

`for (initial ; condition ; step) {action}`
ex: `for(var i=0; i<10; i++) { }`
keywords: `break;` ends loop, `continue;` jumps to the next step (i) inside the loop

`function functionName(arg) {return;}`

use `if(arg == undefined){}` to cover cases where function is called without arg

`function mousePressed()`, `mouseReleased()`, `keyPressed()`, `keyReleased()` define events (outside the draw() function)

button click detection

```
mouseX > button.x
mouseX < button.x + button.width
mouseY > button.y
mouseY < button.y + button.height
```

`translate(x, y)` updates 0,0-point to x,y (`translate(width/2,height/2);` makes center 0)
can be further controlled by `push()` and `pop()`

p5.Vector
initialize with `v1 = createVector(x, y);` (z optional), called `v1.x` or `v1.y`,

Methods:
`v1.rotate(radian)` where `2*PI` is 360° in radian
`v1.mult(n)` multiply with scalar
`v1.normalize()` make vector length = 1

Static methods: `v2 = p5.Vector.mult(v1, 2);` creates new copy

---

p5.sound

HTML:
import `<script src="p5.sound.min.js"></script>` in `<head>` of HTML

preload():

```
soundFormats('mp3','wav');
```

```
mySound = loadSound('path/to/file.wav');
```

```
mySound.setVolume(0.1);
```

To play:

```
mySound.play();
```

Factory pattern

```
function createStuff(input)
{
var stuff = {prop:.., setup: function(){}, draw: function(){}, update:...}
return stuff;
}
```

Constructor

create

```
function MyObjectConstructor(x,y) {
  this.x = x;
  this.y = y;
  this.display = function(scale){ ...}
}
```

- convention: constructor name starts with capital letter
- keyword `this` refers to the new object being created
- we don't have to `return this;`, JS automatically does it

call

```
myObject = new MyObjectConstructor(3,5);
```

keyword `new` creates new object