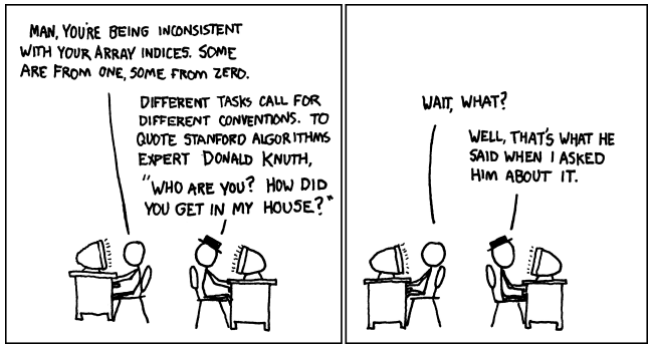
# Impact Lab 2025: Programming Fundamentals
# Lecture 5: Arrays and More Objects

Summer 2025

School of Computing and Data Science

Wentworth Institute of Technology






## Noise Ranges

Using **`map()`** to customize the range of the **`noise()`** output.

```
let t = 0;
function draw(){
  let n = noise(t);
  let x = map(n, 0, 1, 0, width);
  ellipse(x, 180, 16, 16);
  t += 0.01;
}
```

**`map()`** takes five arguments:

- The value you want to map
- The value's current range (min and max)
- The new range you want (min and max)



## Perlin Walker in X and Y

```
class Walker{
  constructor(){
    this.tx = 0;
    this.ty = 10000;
  }
  step(){
    let nx = noise(tx);
    let ny = noise(ty);
    let x = map(nx, 0, 1, 0, width);
    let y = map(ny, 0, 1, 0, height);
    tx += 0.01;
    ty += 0.01;
  }
```

If we let **`tx`** and **`ty`** both start at 0, the object will only move diagonally, so we set **`ty`** to some large value far away from **`tx`**. See the next slide!

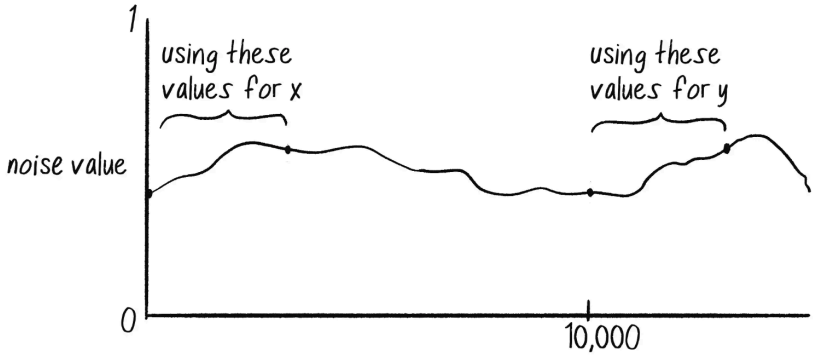


## Perlin Walker in X and Y

```
class Walker{
  constructor(){
    this.tx = 0;
    this.ty = 10000;
  }
  step(){
    let nx = noise(tx);
    let ny
    let x
    let y
    tx +=
    ty +=
  }
```

If we let **`tx`** and **`ty`** both start at 0, the object will only move diagonally, so we set **`ty`** to some large value far away from **`tx`**. See the next slide!

## Perlin Walker in X and Y




Starting **`ty`** at 10000 ensures that we won't overlap with **`tx`**.



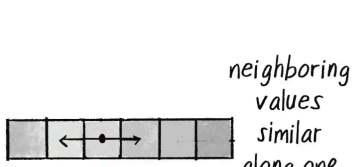
## Exercise

- Take the Perlin Walker and map the result of noise() to the walker's step size instead of its position.

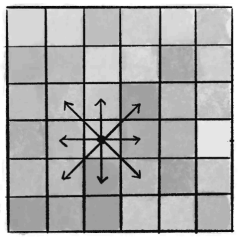


## 2D Noise

- In 1D noise, the sequence of values are similar to their 2 neighbors.
- 2D noise has more neighbors, but the same holds true in both dimensions:



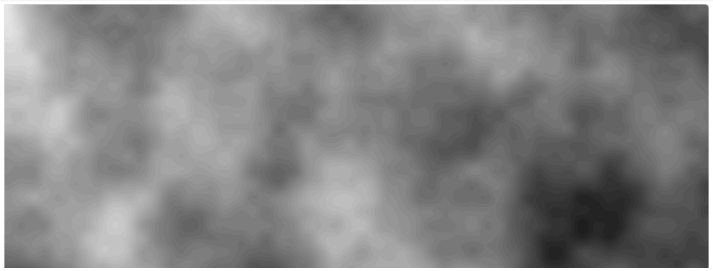
neighboring values similar along one dimension



neighboring values similar along two dimensions



## 2D Noise



2D noise essentially creates an image, it fills a grid of cells with noise values.

To make this visualization, we can just display a color depending on the value of the noise.

## 2D Noise: Visualization

```
loadPixels();
for (let x = 0; x < width; x++) {
  for (let y = 0; y < height; y++) {
    let index = (x + y * width) * 4;
    // A random brightness!
    let bright = random(255);
    // Set the red, green, blue, and alpha values.
    pixels[index] = bright;
    pixels[index + 1] = bright;
    pixels[index + 2] = bright;
    pixels[index + 3] = 255;
  }
}
updatePixels();
```

We loop over all pixels, and give them a brightness (here, it's just the regular **`random()`** call).



## 2D Noise: Visualization

```
loadPixels();
for (let x = 0; x < width; x++) {
  for (let y = 0; y < height; y++) {
    let index = (x + y * width) * 4;
    // A random brightness!
    let bright = random(255);
    // Set the red, green, blue, and alpha values.
    pixels[index] = bright;
    p
    p
    p
  }
}
updatePixels();
```

If you want Perlin brightness instead:

```
let r = noise(x,y);
let bright = map(r, 0, 1, 0, 255);
```

s, and give them a brightness (here, it's just the regular **`random()`** call).



## 2D Noise: Visualization

```
loadPixels();
for
  fo
    pixels[index] = bright;
    p
    p
    p
  }
}
updatePixels();
```

The problem with this version is that the (**x**,**y**) are incremented by 1, which is too large for the step. Let's try and fix it.

If you want Perlin brightness instead:

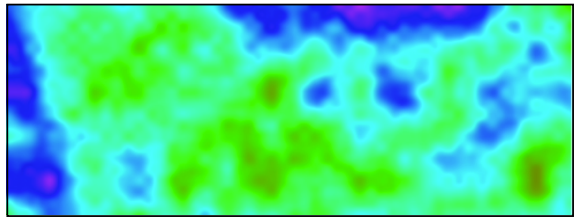
```
let r = noise(x,y);
let bright = map(r, 0, 1, 0, 255);
```

s, and give them a brightness (here, it's just the regular **`random()`** call).



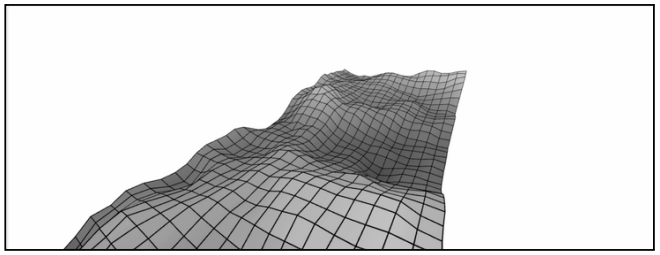
## Exercise

- Play with color to see if you can get an interesting look.
- Add a third argument to you **`noise()`** calls to animate the noise effect.



## Example

- Use the noise values as an elevation of a landscape.



Perlin noise has been used to create landscapes for many movies and video games!



## Topics for Today

- **Arrays**
- Arrays and Objects
- Object Communication



## What is an Array?

An **array** is a list of variables of the same type that represents a set of related values

What if we need to keep track of 1000 items? → Use 1000 variables!
`num1, num2, ...`

Actually, you use an **array**



## Creating an Array

In JavaScript, an **array** is an object, but it also has some new syntax.

The new special symbols, used to denote an array are square brackets: **[ ]**

The idea is to create a collection of variables all of the same type in just one step:

```
let myArray=[];
let anotherArray=[1,2,3,4,5];
let yetAnotherArray=["hello","world"];
```

## Creating an Array

```
let myArray=[];
let anotherArray=[1,2,3,4,5];
let yetAnotherArray=["hello","world"];
```

Start by creating a regular variable with **let** and seeing it equal to **[]**

This new array can hold as many elements as you want, including predefined values.



## A Collection of Variables

You can think of creating an array as declaring many individual variables:

```
let counts=[];
```

This is similar, but not exactly the same as declaring many separate variables:

```
let counts0, counts1, counts2, counts3, counts4, counts5, counts6, counts7;
```



## Accessing Array Elements

To actually use an individual element in the array, you must specify the **index** of the element in brackets

Be careful not to confuse the two uses of brackets (array creation versus array use)

**Example:** Array named counts. Setting the value at index 7 to 10:

```
// create an array
let counts=[];
// assign element 7 a value of 10
counts[7] = 10;
```



## Arrays in Memory

Arrays are stored in memory so that all the elements in the array are sequential

```
let counts=[0,0,0,0,0,0,0,0];
counts[3] = 10;
```

| address | value | variable |
|---|---|---|
| 1000 | 0 | counts[0] |
| 1004 | 0 | counts[1] |
| 1008 | 0 | counts[2] |
| 1012 | 10 | counts[3] |
| 1016 | 0 | counts[4] |
| 1020 | 0 | counts[5] |
| 1024 | 0 | counts[6] |
| 1028 | 0 | counts[7] |
| 1032 | | |
| 1036 | | |

## Array Elements and Length

**IMPORTANT NOTE:** Arrays start at index **0** and go through index **`size-1`**

Use **ARRAY.length** to get the size of the array

Arrays do **NOT** start at index 1!

Array indices do not have to be hard coded, they can be any expression that evaluates to an integer:

```
let temperatures = [];
for (let i = 0; i < 64; i++) {
  temperatures[i] = 50;
}
```

What does this code do?



## Out of Bounds

You should always have to ensure that your program only uses valid elements/indices for an array

You shouldn't access an index greater than or equal to the length of the array

For the most part, if you access an index that hasn't been set, it will be evaluated to **`undefined`**.

```
let myArray = [];
myArray[0] = 5;  // ok
myArray[9] = -6; // ok
print(myArray[-1]); // will print undefined
print(myArray[10]); // will print undefined
```



## How Useful is it, Really?

If I have many ellipses to draw, I can use an array to store some of the data:

```
let nums=[100,25,72,45];
function draw(){
  background(0);
  ellipse(100,200,nums[0],nums[0]);
  ellipse(200,200,nums[1],nums[1]);
  ellipse(300,200,nums[2],nums[2]);
  ellipse(400,200,nums[3],nums[3]);
}
```

But, there's still a bunch of repeated code...



## How Useful is it, Really?

Use a loop!

```
let nums=[100,25,72,45];
function draw(){
  background(0);
  for(let i=0;i<4;i++){
      ellipse((i+1)*100,200,nums[i],nums[i]);
  }
}
```

Loops and Arrays are almost always used together

## Topics for Today

- Arrays
- **Arrays and Objects**
- Object Communication



## Let's Go Back to Bubbles!

We've already defined our bubbles and created multiple bubbles.

```
let bubble0;
let bubble1;
let bubble2;
let bubble3;
let bubble4;
```

But...that's a lot of copied code.

And what if I want 1000 bubbles?

Close...but still not great.

```
let bubbles=[];
bubbles[0]=new Bubble(…);
```



## Let's Go Back to Bubbles!

We can use a loop again to create all our bubbles.

```
for(let i=0;i<5;i++){
  bubbles[i]=new Bubble(…);
}
```

Now, I can create as many bubbles as I want!

Depending on the values that we use to initialize the bubble, we may need an equation to determine positions, sizes, etc.



## Adding Bubbles

What if we wanted to create a bubble every time we clicked the mouse?

Arrays are actually objects too! They have many features that allow us to add/remove/manipulate elements of an array.

```
function mousePressed(){
  let size=random(10,50);
  let b = new Bubble(mouseX,mouseY,size);
  bubbles.push(b);
}
```

Change **mousePressed()** to **mouseDragged()**

Adds **b** to the next available space in the **bubbles** array.

## Mouse Interaction

We have a function called **`mousePressed()`** that p5.js can use to check if we click the mouse.

Unfortunately, p5.js can't tell we click on a shape, even with **`mousePressed()`**.

We have to come up with our own equation to determine if we've clicked on an object.

We need some code to tell us a point we click on is within a circle...



## A Quick Note on Style

When creating classes, it's useful to limit what external information a class needs.

For example, our bubble class needs to know about **`mouseX`** and **`mouseY`**.

This is called a dependency.

What if we decided to change to a different graphics library that didn't have **`mouseX`** and **`mouseY`**?

We should write our classes in such a way that they have the fewest dependencies possible.

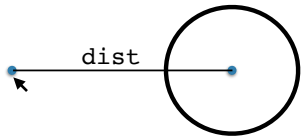


## Bubbles Exercise

Let's play with the **`Bubble`** class and mouse interaction.

p5.js has a convenient function called **`dist()`** that takes the **`x`** and **`y`** position of two objects.

Why do you think this is useful?




We can compute the distance between the mouse and the center of the circle.

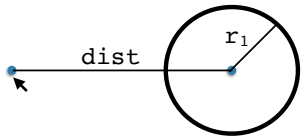


## Bubbles Exercise

- Since we know the radius of the circle and the distance between the mouse and the center of the circle:

The mouse is in the circle if the distance is less than the radius!

How might we change the bubble if we just hover the mouse over it without clicking?




So, we can add this calculation to **`mousePressed()`** to do things when we click the bubble.

## Removing Elements from an Array

Maybe we want to pop our bubbles when we click.

There are several ways to remove elements in an array depending on what you want to remove.

**`pop()`** removes the last element

**`splice()`** removes at a selected index

If we know the index of the bubble in the array, we can just remove it so it no longer draws.

Let's play around with removing elements. We need to take care in certain situations.

## Removing Elements from an Array

```
function mousePressed() {
  for (let i = 0; i < bubbles.length; i++) {
    let x = bubbles[i].x;
    let y = bubbles[i].y;
    let d = dist(x, y, mouseX, mouseY);
    if (d < bubbles[i].r) {
      bubbles.splice(i,1);
      i--;
    }
  }
}
```

Checking all the bubbles

Computing distance to mouse

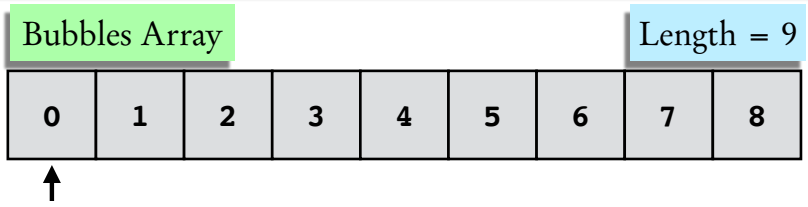
**`splice()`** one bubble at index **`i`**

Simple **`mousePressed()`** function for removing bubbles on click

Why do **`i--`**?



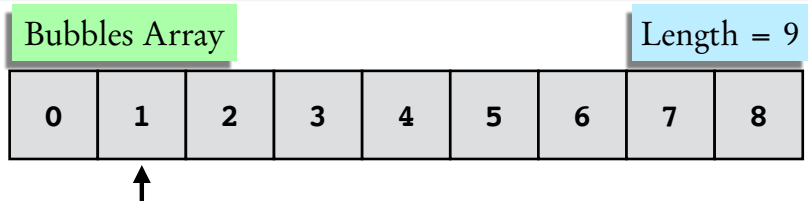
## Removing Elements from an Array




As we loop over the bubbles array…

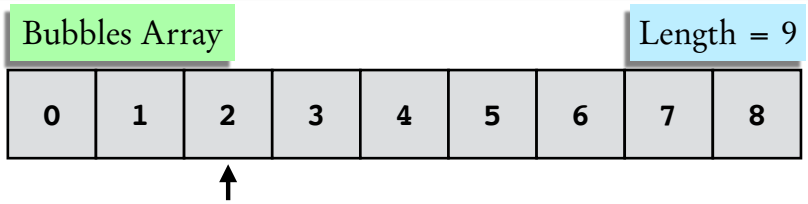


## Removing Elements from an Array




As we loop over the bubbles array…

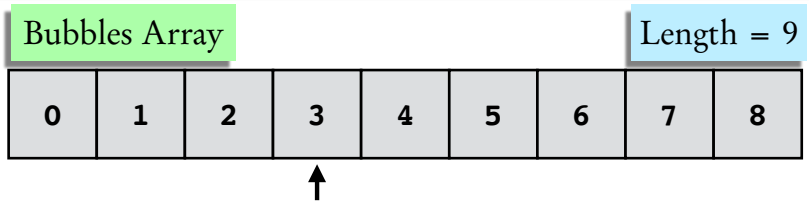
## Removing Elements from an Array




As we loop over the bubbles array...



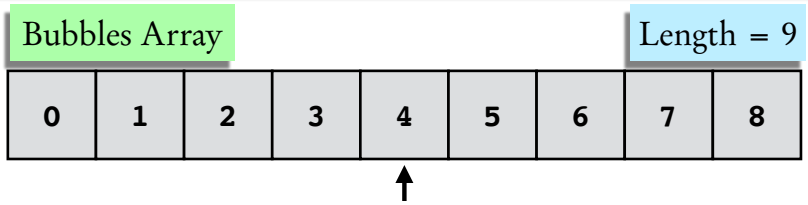
## Removing Elements from an Array




As we loop over the bubbles array...



## Removing Elements from an Array

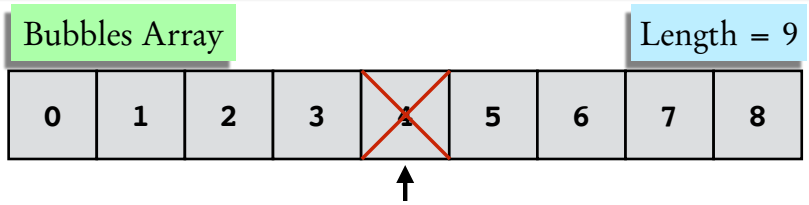



As we loop over the bubbles array...

Until we find one we want to remove...



## Removing Elements from an Array




As we loop over the bubbles array...

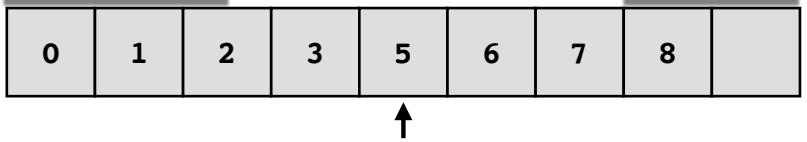
Until we find one we want to remove...

```
bubbles.splice(4,1);
```

## Removing Elements from an Array

Bubbles Array

Length = 8




As we loop over the bubbles array...

Until we find one we want to remove...

```
bubbles.splice(4,1);
```

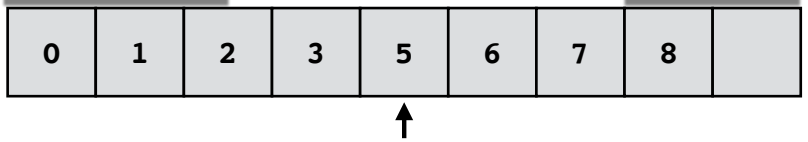
Arrays in p5.js automatically adjust when you remove and element, so now element 5 is where 4 used to be.



## Removing Elements from an Array

Bubbles Array

Length = 8




If we don't do the `i--`, we would skip checking 5 because we would move to 6 on the next iteration.

Arrays in p5.js automatically adjust when you remove and element, so now element 5 is where 4 used to be.



## Topics for Today

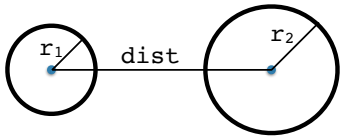
- Arrays
- Arrays and Objects
- **Object Communication**



## Object Communication

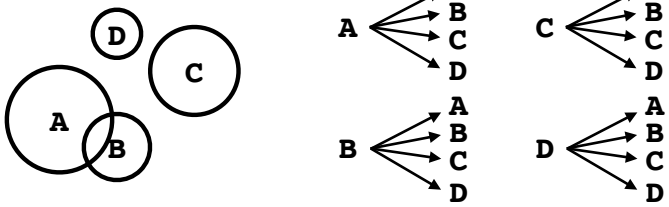
We can already tell if an object and the mouse interact, but what about multiple objects?

For circles, it's nice and easy, but for other objects it can be very complicated.



# More Nested Loops

If I have many bubbles, I need to check every pairwise combination of bubbles.










So, we pick a bubble, loop though every other bubble, and check if they touch.