programming for artists and designers

Daniel Berio

Session Summary

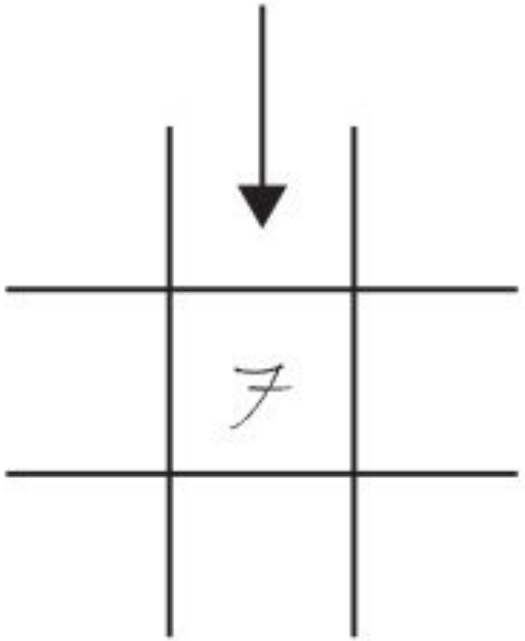
- Mid-term Q&A
- Arrays
- Vectors
  - 2d vector math basics
  - glm::vec2
- Number Generators:
  - Interpolation
  - Perlin Noise
  - Flow fields
- Lab / at home:
  - Chance & Control: Generative “Drawing”

Mid-term Q&A

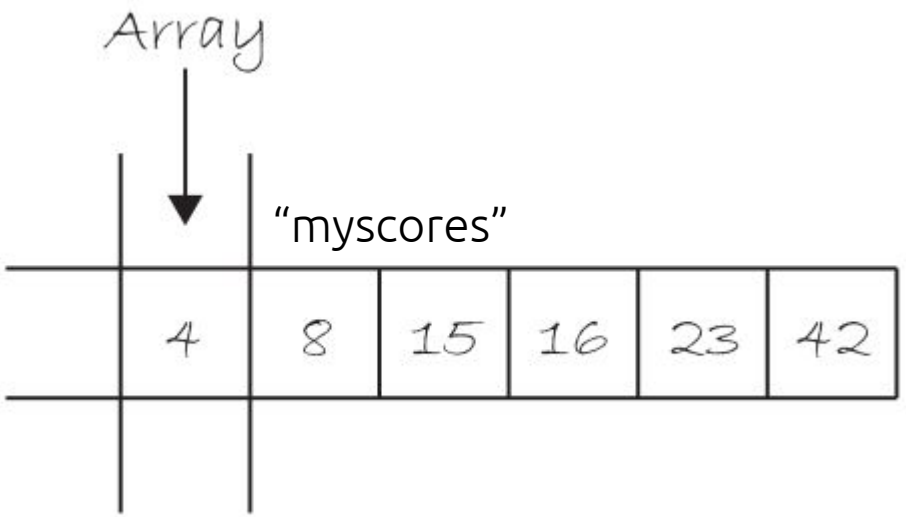
Any questions about the project brief?

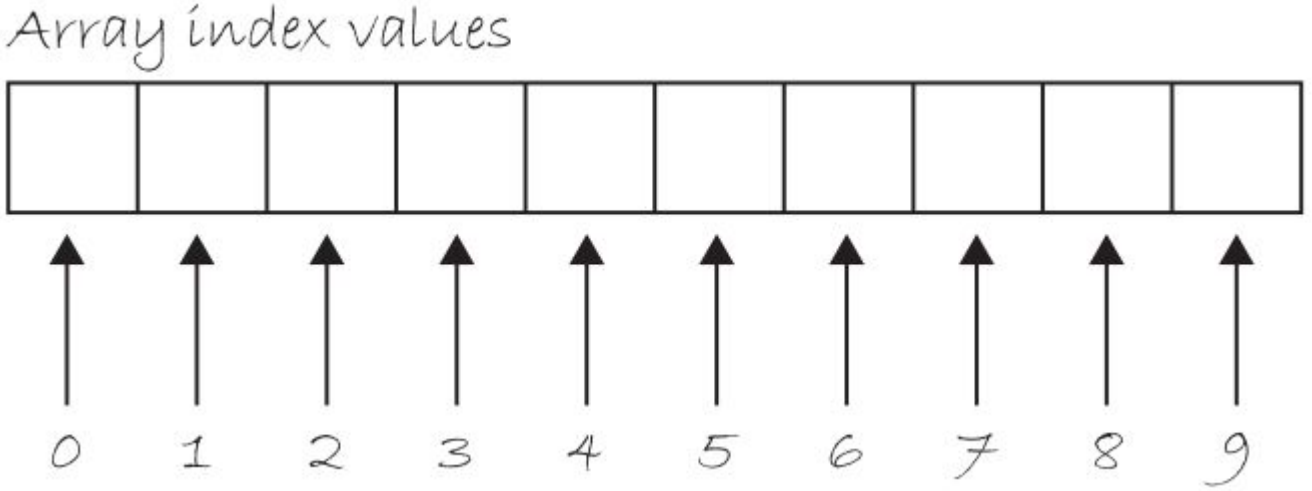
How do we store a list of values?

```
int myscore;
```



Arrays

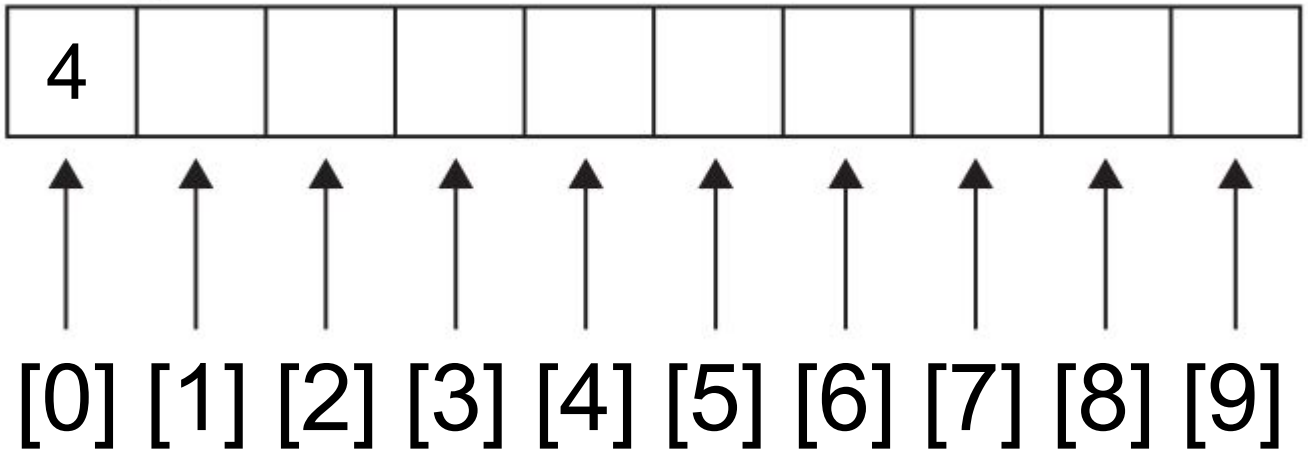




Storing a value inside an array

```
//in ofApp.h
int myscores[10];

//in setup()
myscores[0] = 4;
```



Storing a value inside an array

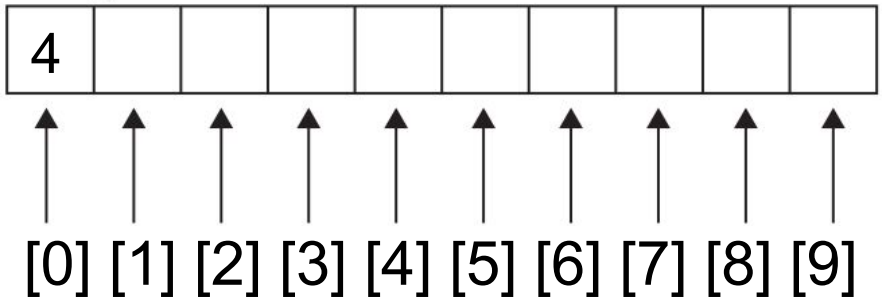
With C++, arrays are blocks of *static computer memory* whose size must be known at compile time, before the program runs.

```
//in ofApp.h
int myscores[10];
```

```
//in setup()
myscores[0] = 4;
```

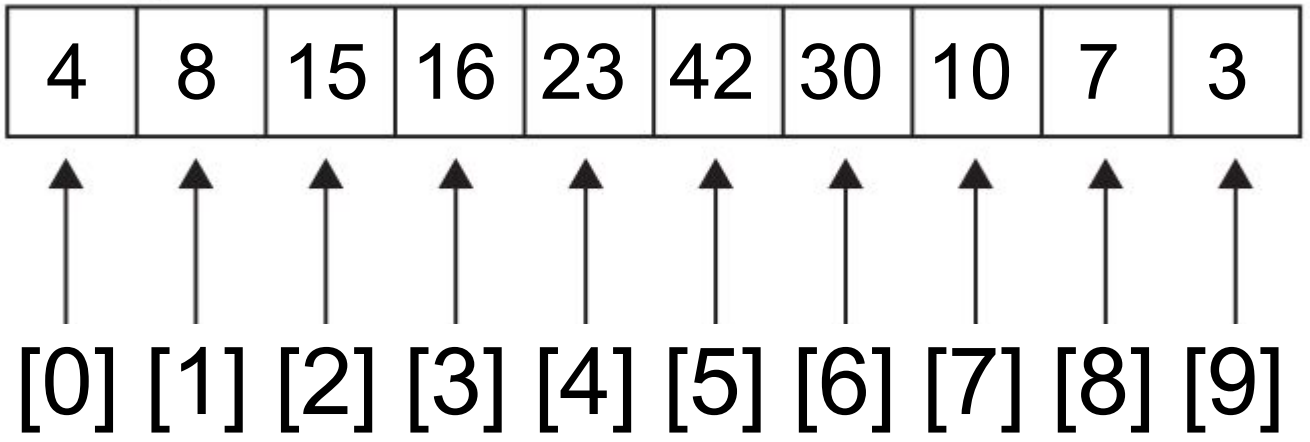
Different from Javascript (p5.js) !!!!!!

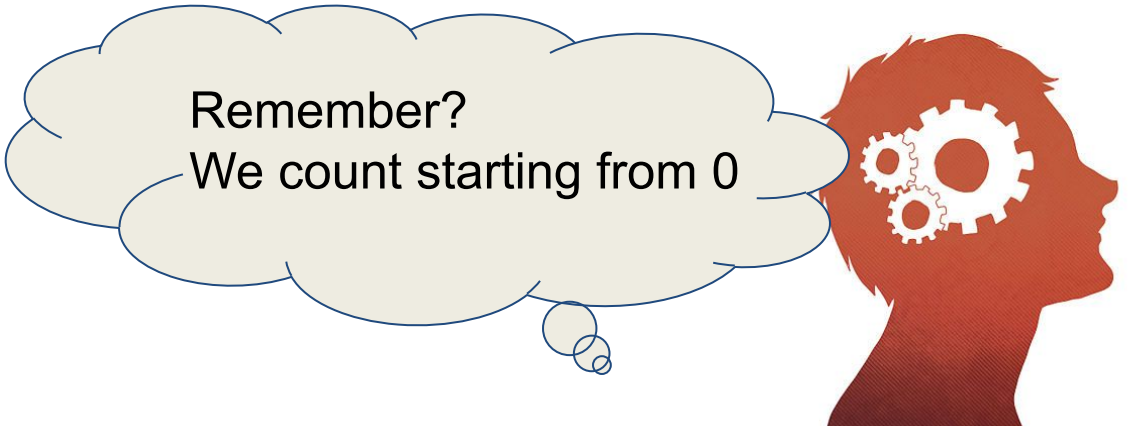
Next week we will learn about a data structure that we can *dynamically* make bigger while the program is running.



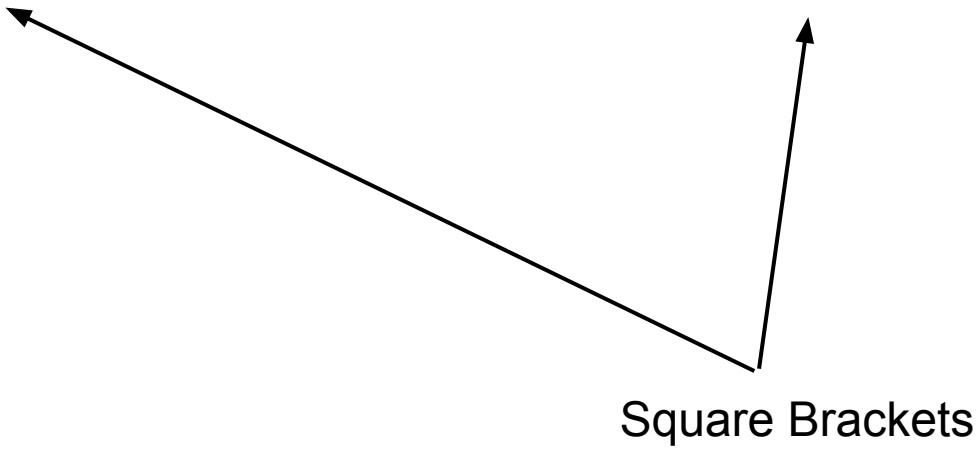
Reading a value inside an array

```
cout << myscores[4] << endl;
```





myArray[arrayIndexValue]



Question: What do we do if we have an array with 1000 elements?

```
//in ofApp.h
int myIntArray[1000];

//in setup()
myIntArray[0] = 10;
myIntArray[1] = 80;
myIntArray[2] = 5;
…
myIntArray[999] = 100;
```

Answer

We can use a for loop!!!!

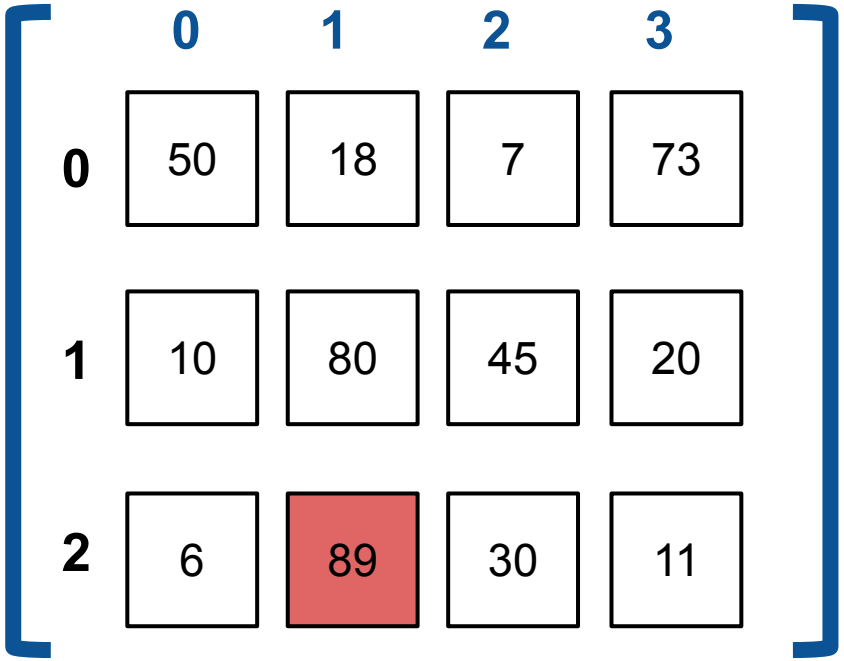
```
//in ofApp.h
int myIntArray[1000];

//in setup()
for (int i = 0; i<1000; i++)
{
   //1000 random values
   myIntArray[i] = ofRandom(0,10);
}
```

2D Arrays (arrays of arrays)

```
int array2D[3][4];
```

```
cout << array2D[2][1] << endl;
```



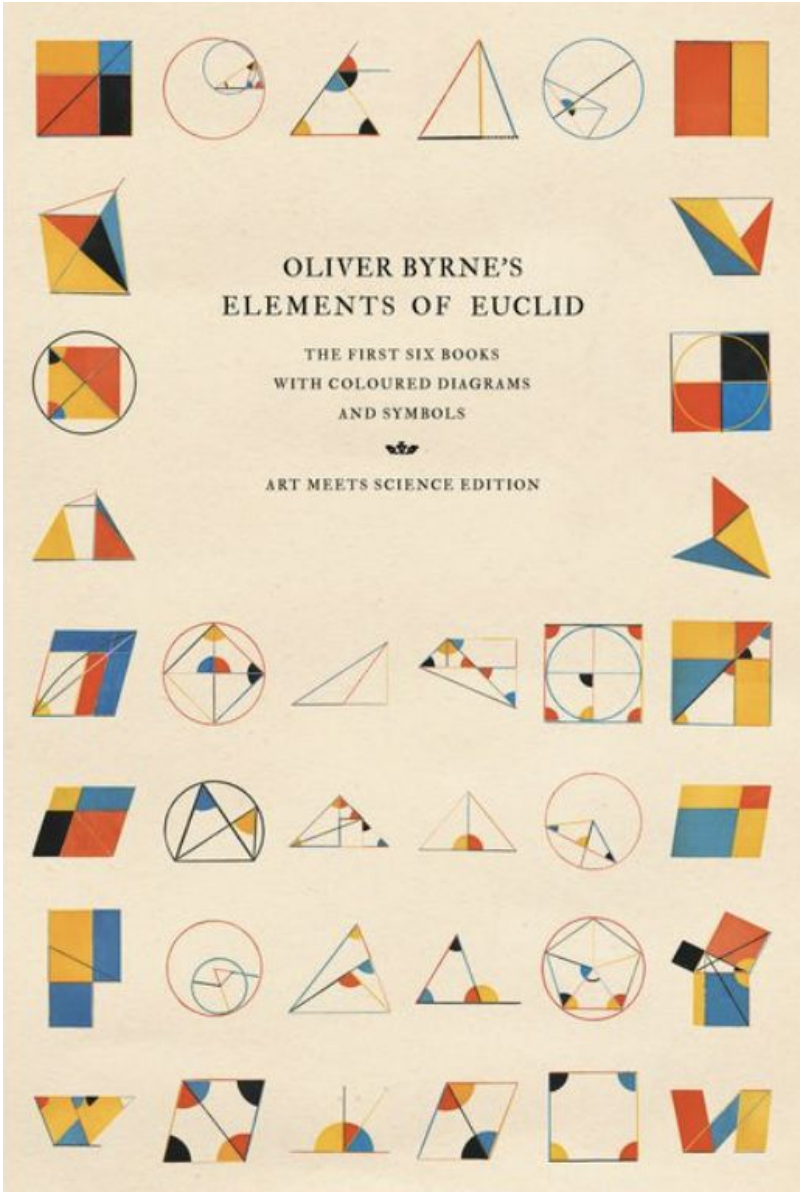
Quick way to fill a 2D array?

A nested for loop!

```
//in ofApp.h
float myFloatArray[10][10];

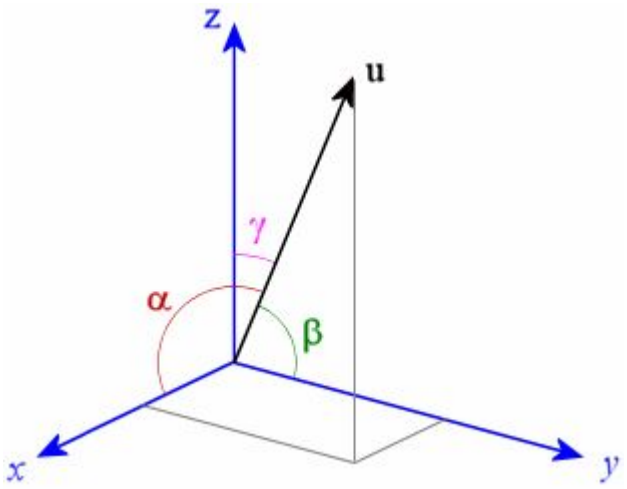
//in setup()
for (int i = 0; i<10; i++)
{
    for (int j = 0; j<10; j++)
    {
        //10 x 10 (100) random values between 0-1
        myFloatArray[i][j] = ofRandom(1);
    }
}
```

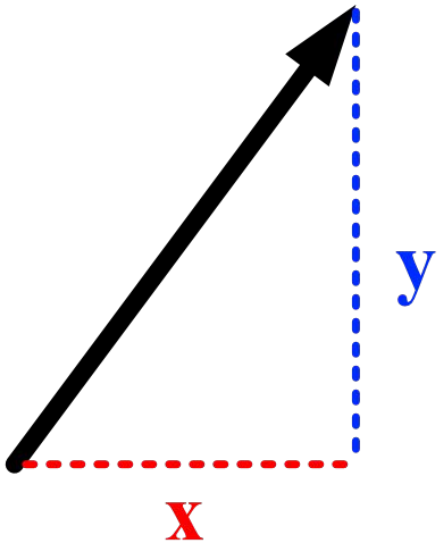
Back to geometry



Vectors

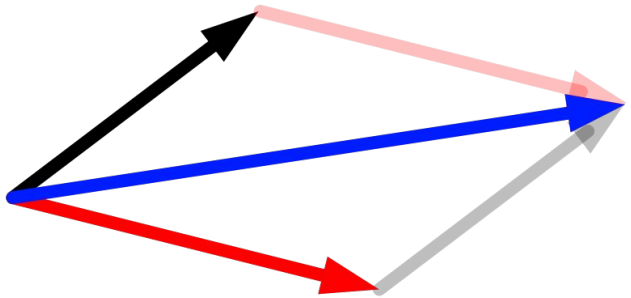
- Vectors are mathematical objects that describe a **direction** and a **length** (aka “magnitude” or “norm”)
- Imagine them as arrows in space (e.g. 2D, 3D, etc…)
  - Tip of the arrow pointing in the direction of the vector
- Represented with an ordered list of numbers
  - One for each dimension, e.g. 2 for 2D, 3 for 3D, 4 for 4D etc etc…
  - We will focus only on 2D, i.e. [x, y]
  - Concepts also useful in 3D and further higher dimensions (e.g. machine learning)
- Similar to points but (conceptually) vectors don’t have a fixed position
  - Useful to encode quantities like displacement, velocity, a force





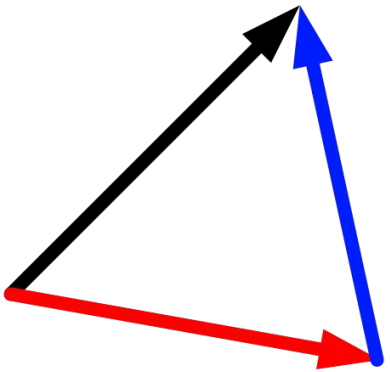
Vector math

- Vectors can be manipulated with operations such as:
  - Multiplication/division by a number -> scaling
  - Addition/subtraction
  - Other operations that we will not cover
    - E.g. dot product , cross/wedge product
  - Element-wise multiplication, but no intuitive geometric interpretation



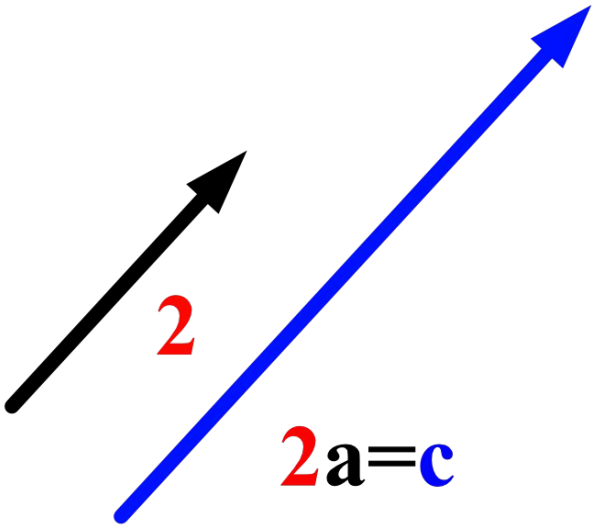
a+b=c

"Parallelogram rule"



a-b=c

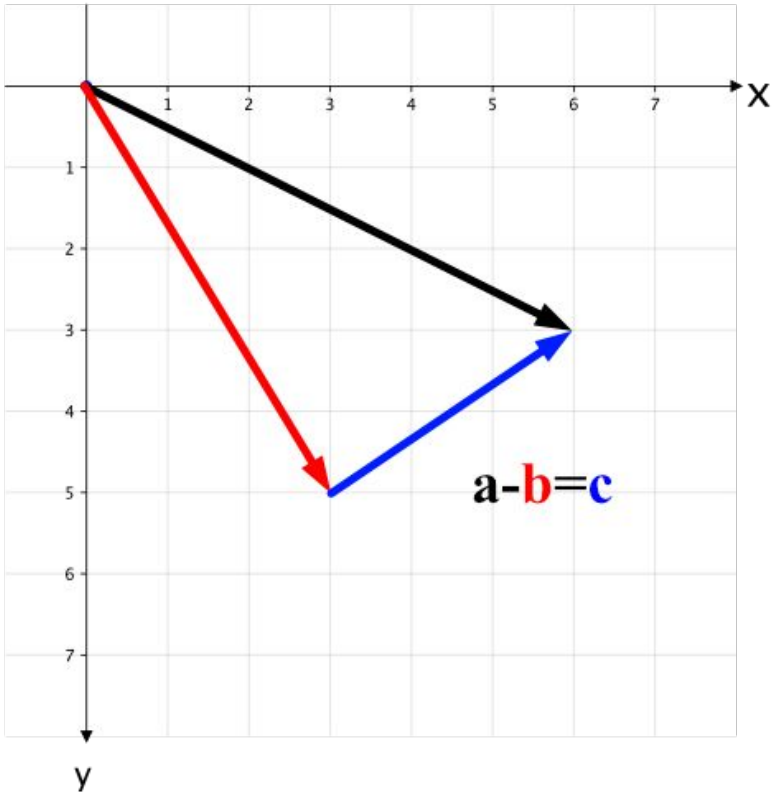
Vector taking me from **b** to **a**



Scaling/weighting
"How much of the vector"

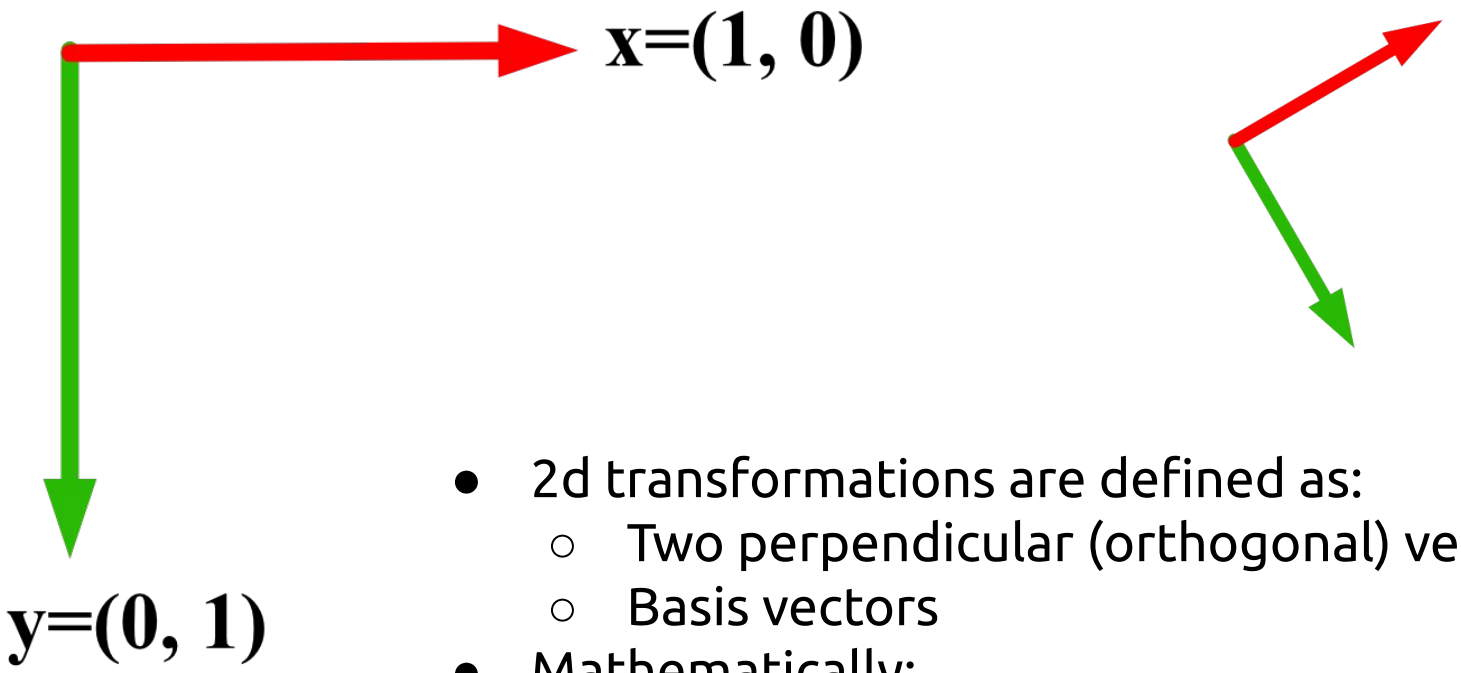
Vector math

- Vectors and positions are **conceptually** different
- But with a coordinate system we can encode positions as vectors
  - Offset from the origin
  - Difference between two positions (encoded as vectors) gives us a vector
    - Displacement from the first position to the second



We have actually seen vectors since Week1!

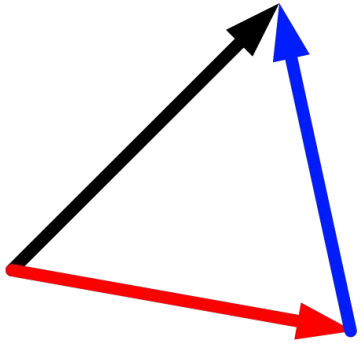
Coordinate systems + transformations



- 2d transformations are defined as:
  - Two perpendicular (orthogonal) vectors and an origin
  - Basis vectors
- Mathematically:
  - Describe a point as a linear combination of basis vectors
    - E.g. $\mathbf{p} = 2\mathbf{x} + 4\mathbf{y}$
    - Try on paper (parallelogram rule)
  - Organized as 3 columns of a “matrix”
  - A matrix can transform a vector
  - That is why we use the term “ofPushMatrix”

We have actually used vectors in Week 2!

(gestural drawing)

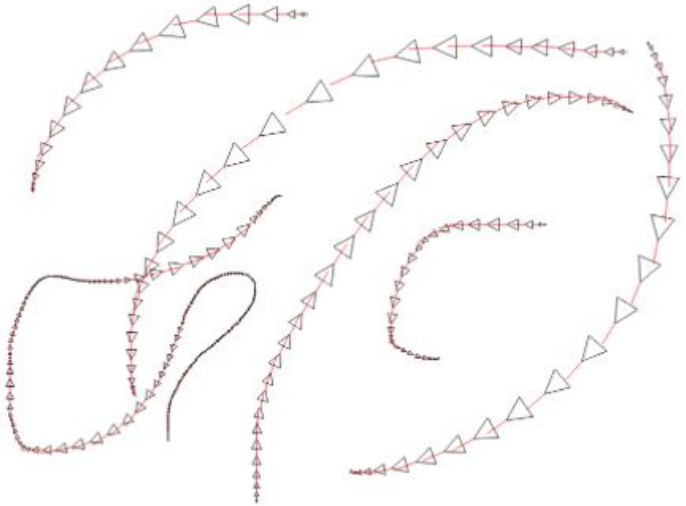


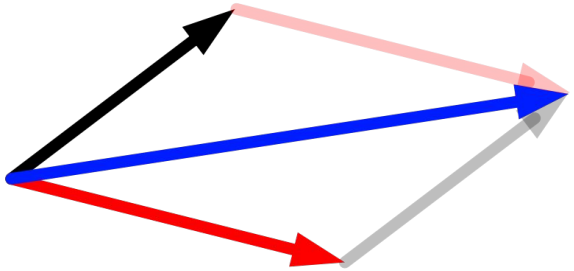
d = mouse - cur

```
void ofApp::draw(){

    float dx = (mouseX - curx)*amt;
    float dy = (mouseY - cury)*amt;

    curx = curx + dx;
    cury = cury + dy;
```

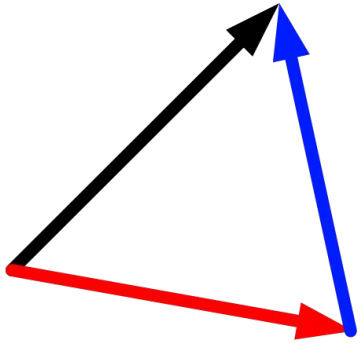




cur = cur + d

We have actually used vectors in Week 2!

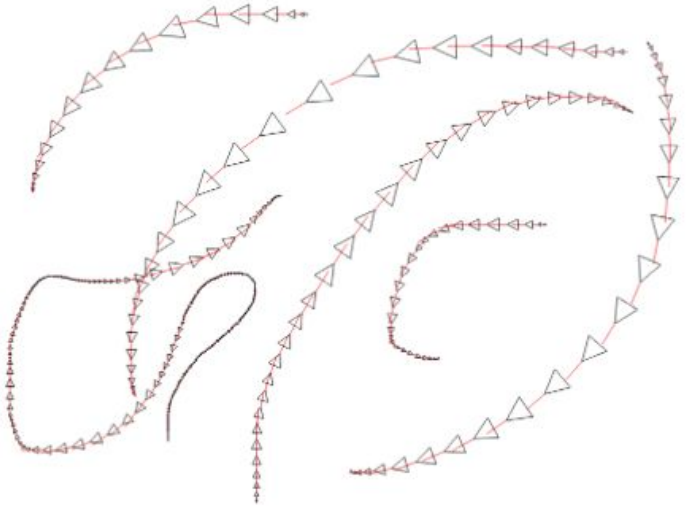
(gestural drawing)

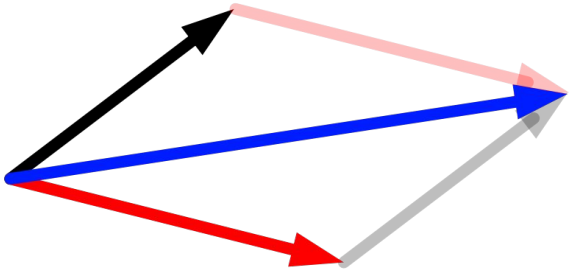


d = mouse - cur

```
void ofApp::draw(){

    vec2 d = (mouse - cur)*amt;
    cur = cur + d;
```

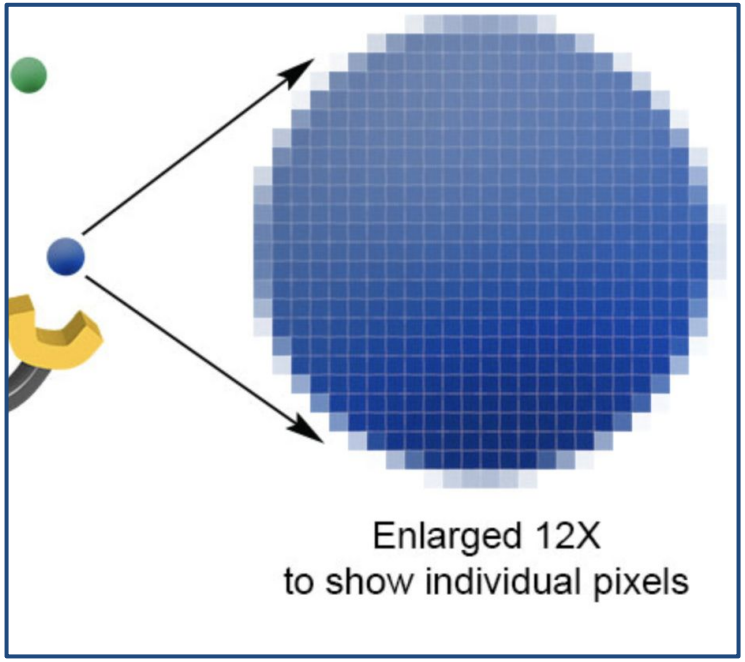




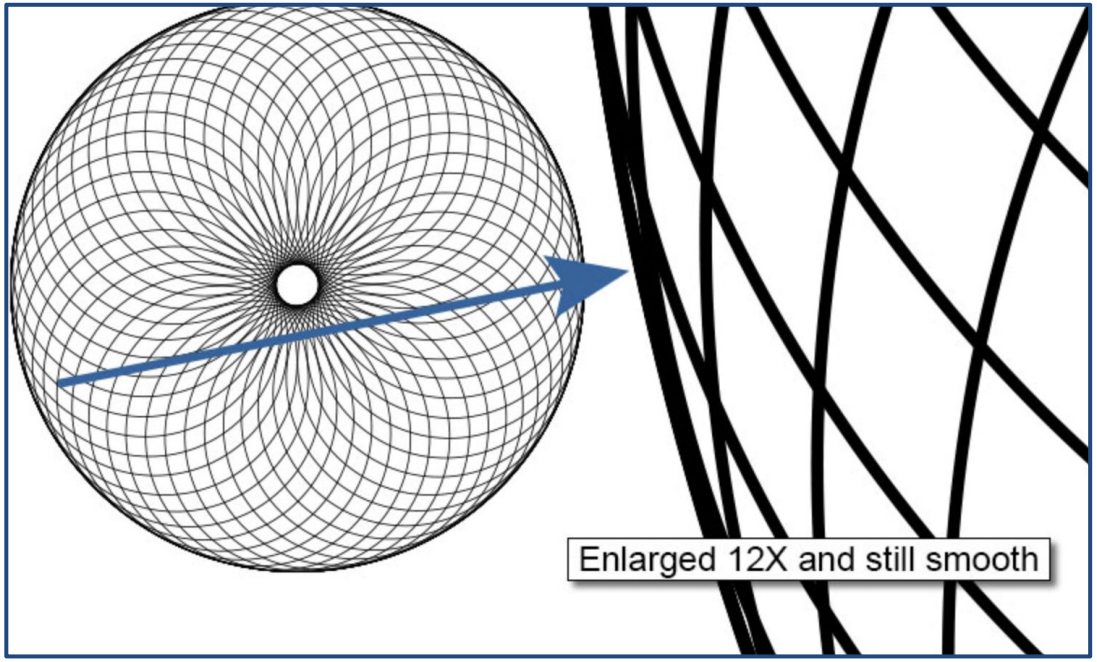
cur = cur + d

OUTPUT

Bitmap vs. Vector Graphics



Bitmap enlarged - we see pixels



Vector graphics - scale better as information is stored as xy coordinates & styling

outputExamples

glm::vec2

we are doing this

In header file ofApp.h or at beginning of ofApp.cpp :

```
using namespace glm;
```

In ofApp.cpp :

```
vec2 myVector;
```

Without “using namespace” we would write:

```
glm::vec2 myVector;
```

https://openframeworks.cc/documentation/glm/
https://openframeworks.cc///documentation/glm/glm::vec2/

We have seen this before!

In week 2 helloWorld.cpp :

```
#include <iostream>
using namespace std;

int main()
{
   cout << "Hello World!" << endl;
}
```

glm::vec2

We can store x and y coordinates in a vec2, e.g.

```
vec2 myVector = vec2(mouseX,mouseY);
myVector.x = 100; //change value
```

https://openframeworks.cc/documentation/glm/
https://openframeworks.cc///documentation/glm/glm::vec2/

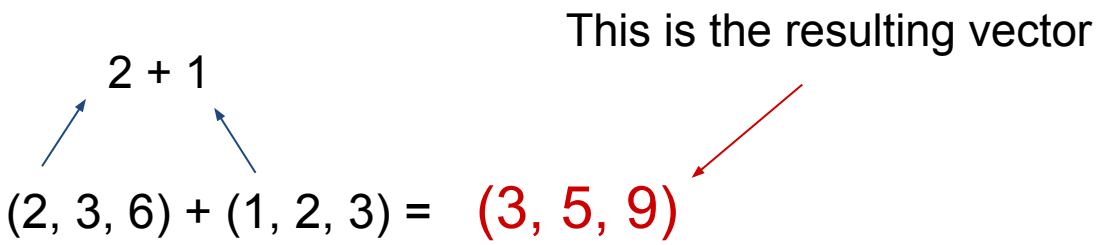
vec2 is for two values – xy

- Dot notation
  - `vec2` is a C++ (class) type
  - Initialize with a “constructor” (like a function call):
    - `vec2 myVec = vec2(mouseX, mouseY);`
  - `vec2.x` gives us a access to the **x** component
  - `vec2.y` gives us a access to the **y** component
  - Can set a component, e.g:
    - ```
      vec2 myVec = vec2(3, 4);
      myVec.x = 1.0;
      ```
  - You will see in function definitions that take a vector:
    - `void ofTranslate(const glm::vec2& p)`
    - For the moment don’t worry about it!
      - Just think, it takes a `vec2` and does not modify it

https://openframeworks.cc///documentation/glm/detail_func_geometric/
https://natureofcode.com/book/chapter-1-vectors/
      ```

Addition, Subtraction, Multiplication & Division

Addition and Subtraction are an element-wise operation:



Without vectors (error prone)

```
float ax=2.0, ay=3.0;
float bx=1.0, by=2.0;
float cx = ax + bx;
float cy = ay + by;
```

vs

With vectors

```
vec2 a = vec2(2.0, 3.0);
vec2 b = vec2(1.0, 2.0);
vec2 c = a + b;
```

https://natureofcode.com/book/chapter-1-vectors/

Addition, Subtraction, Multiplication & Division

Multiplication and Division are done by multiplying/dividing each element by a scalar:

Scalar

(2, 3, 6) * 2 = (4, 6, 12)

Without vectors (error prone)

```
float ax=2.0, ay=3.0;
float bx = ax*4;
float by = ay*4;
```

vs

With vectors

```
vec2 a = vec2(2.0, 3.0);
vec2 b = a*4;
```

or

```
vec2 b = vec2(2.0, 3.0)*4;
```

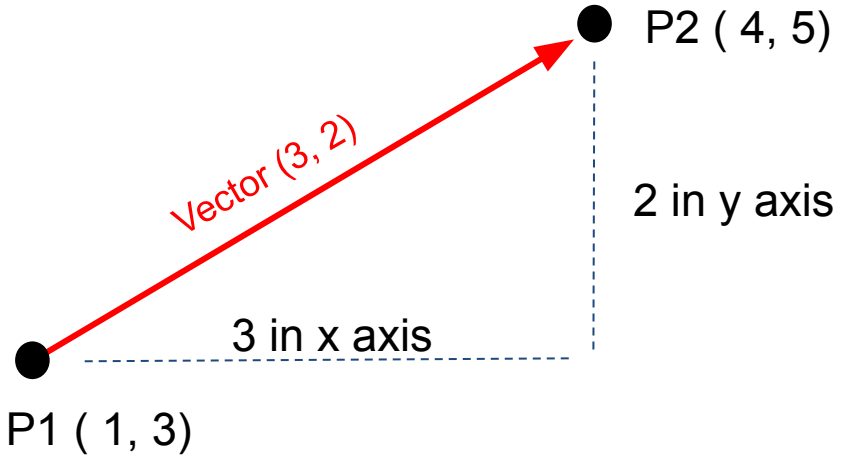
https://natureofcode.com/book/chapter-1-vectors/

VectorMath

vec2 is for two values – xy

*It has a **Length** or **Magnitude***
(i.e. how long is this red arrow?)

```
vec2 redArrow = vec2(3,2);
float mag = length(redArrow);
```



*It has a **Direction***
(i.e. what direction is the arrow pointing?)

```
vec2 redArrow = vec2(3,2);
```

Normalizing a vector makes it a unit vector

```
float angle = atan2(dir.y, dir.x); // or redArrow
vec2 dir = normalize(redArrow);
```

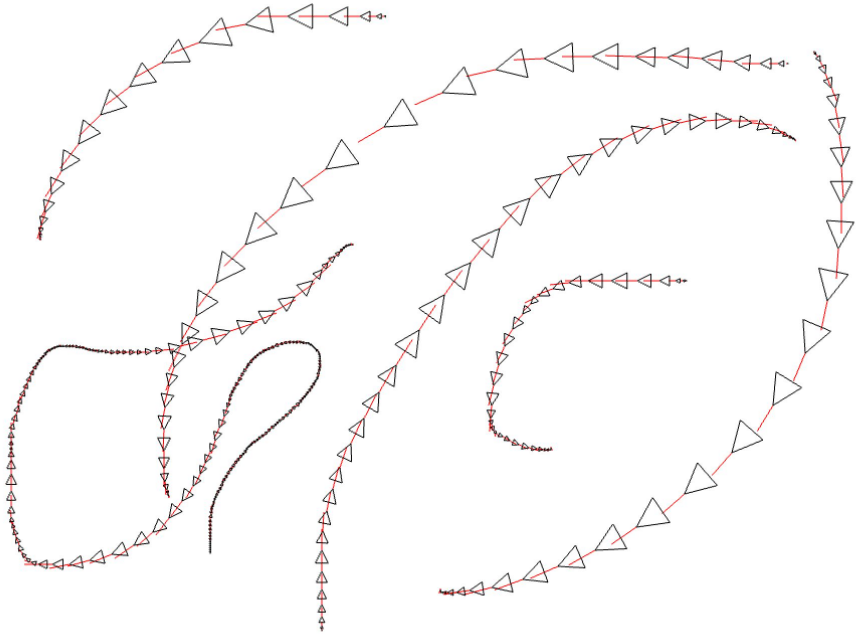
A **unit** vector is a vector of length 1:

```
vec2 dir = vec2(cos(angle), sin(angle))
```

https://openframeworks.cc///documentation/glm/detail_func_geometric/
https://natureofcode.com/book/chapter-1-vectors/

control

mouse speed gives us gestural control



Output like this can be achieved using the Gestural Drawing Activity

easing_Vec2

Break?

Export one “frame” of drawing as SVG

In ofApp.h:

```
bool exportVectorGraphics; //set to false in setup()
```

In draw():

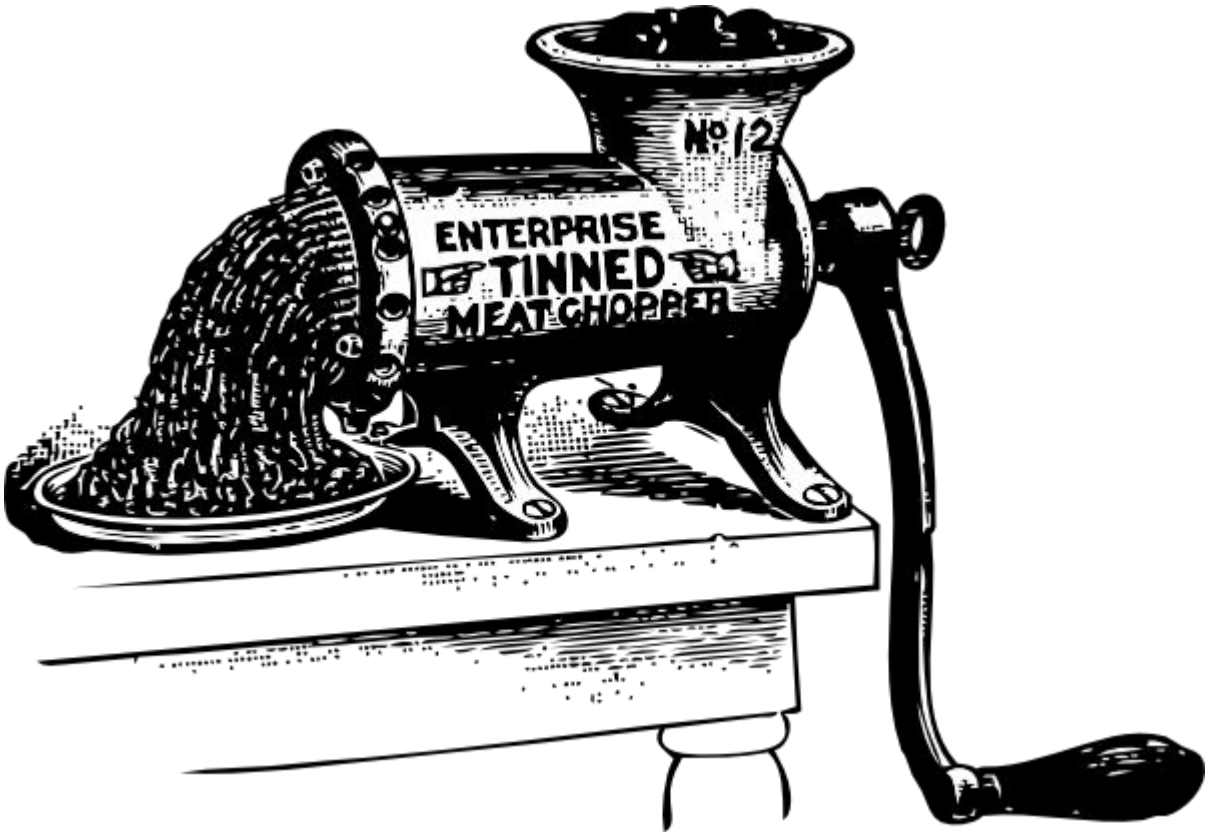
```
if (exportVectorGraphics){
   ofBeginSaveScreenAsSVG(“nameOfFile.svg”);
}
//our drawing code
if(exportVectorGraphics){
    ofEndSaveScreenAsSVG();
    exportVectorGraphics = false;
}
```

In keyPressed():

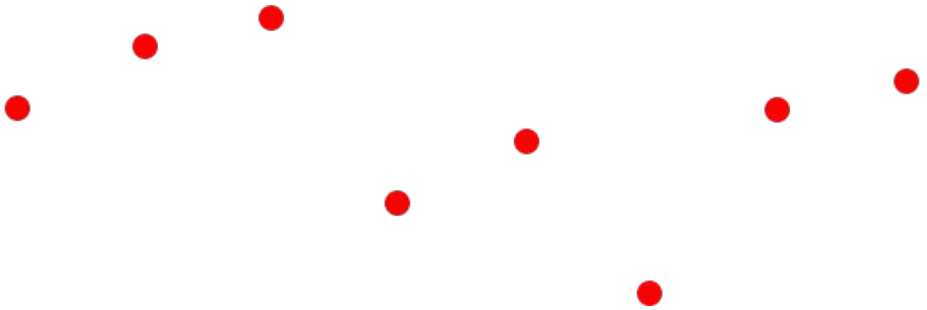
```
exportVectorGraphics = true;
```

Break?

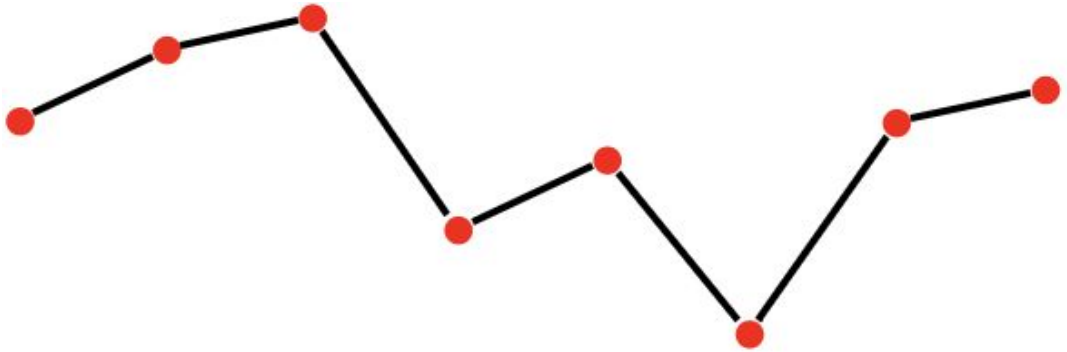
Smooth randomness



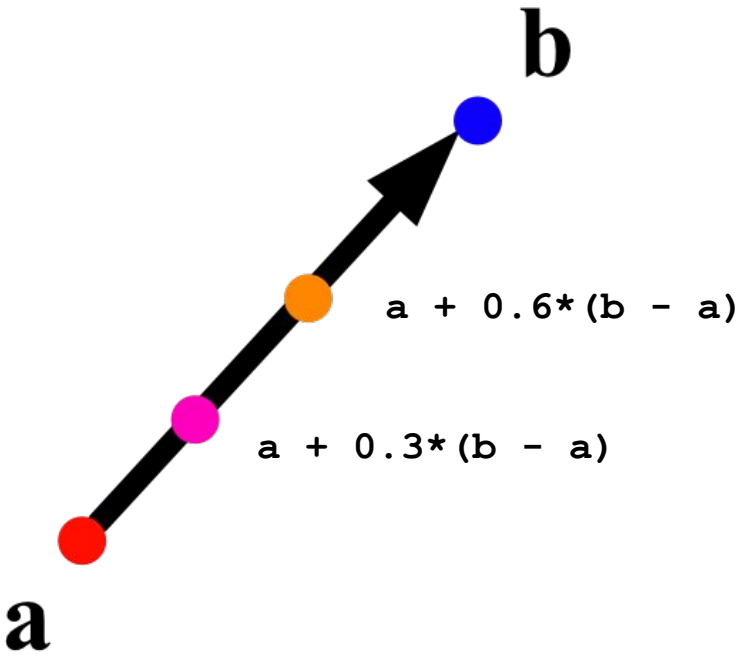
interpolation



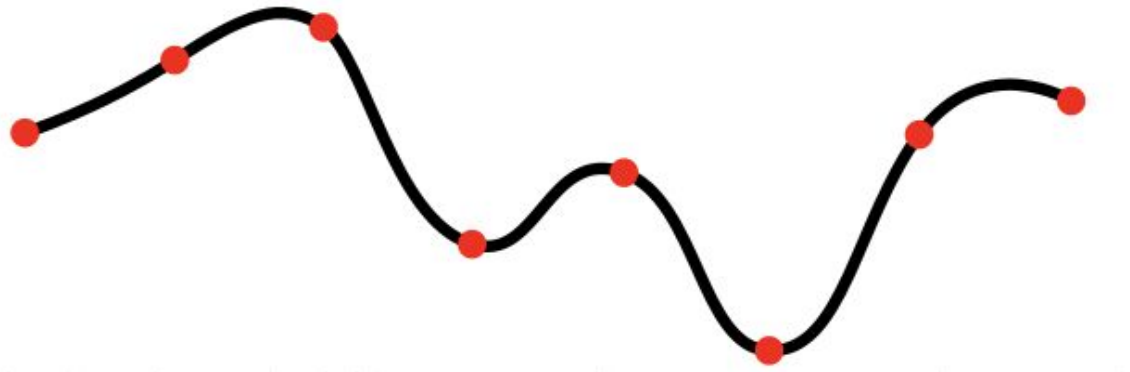
Input points (discrete)



Linear interpolation



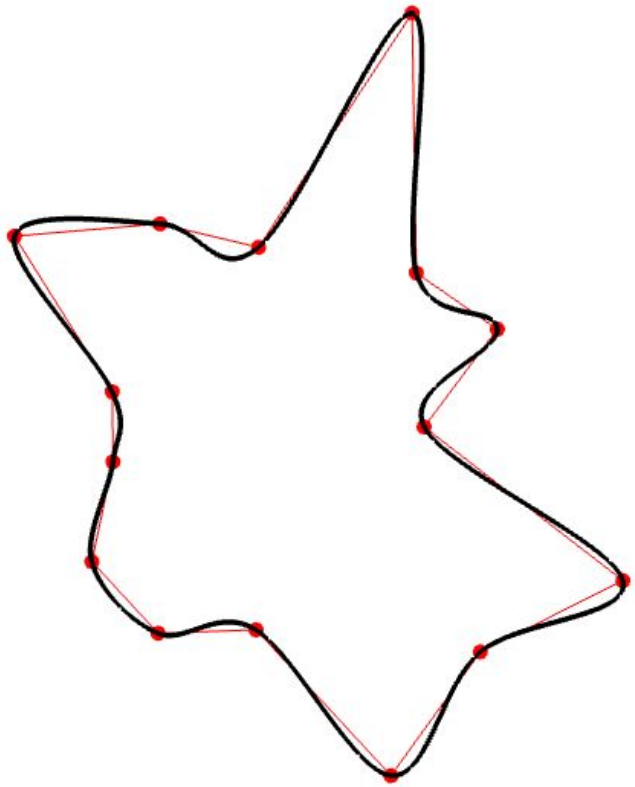
Linear (weighted sum)

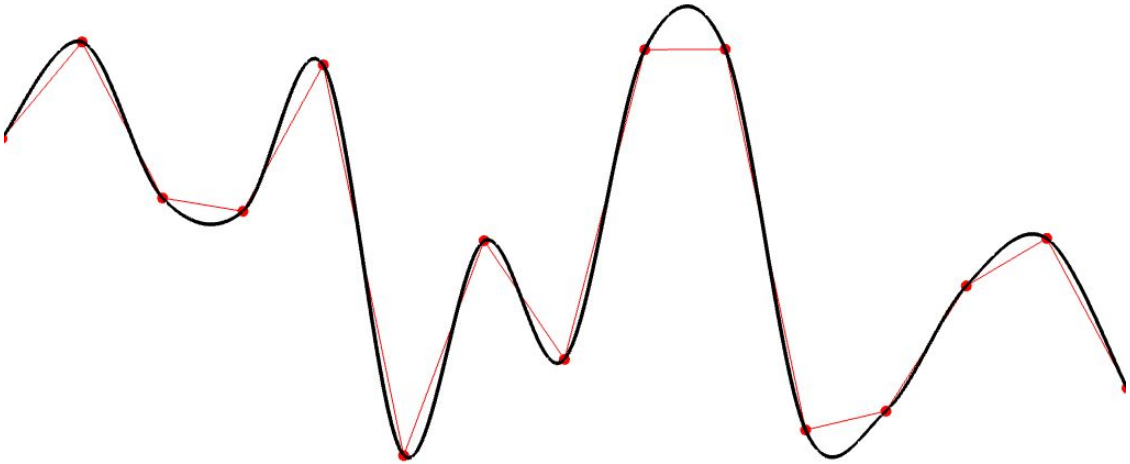


Spline interpolation (cubic)

https://en.wikipedia.org/wiki/Interpolation

ofRandom + interpolation
(with ofCurveVertex)





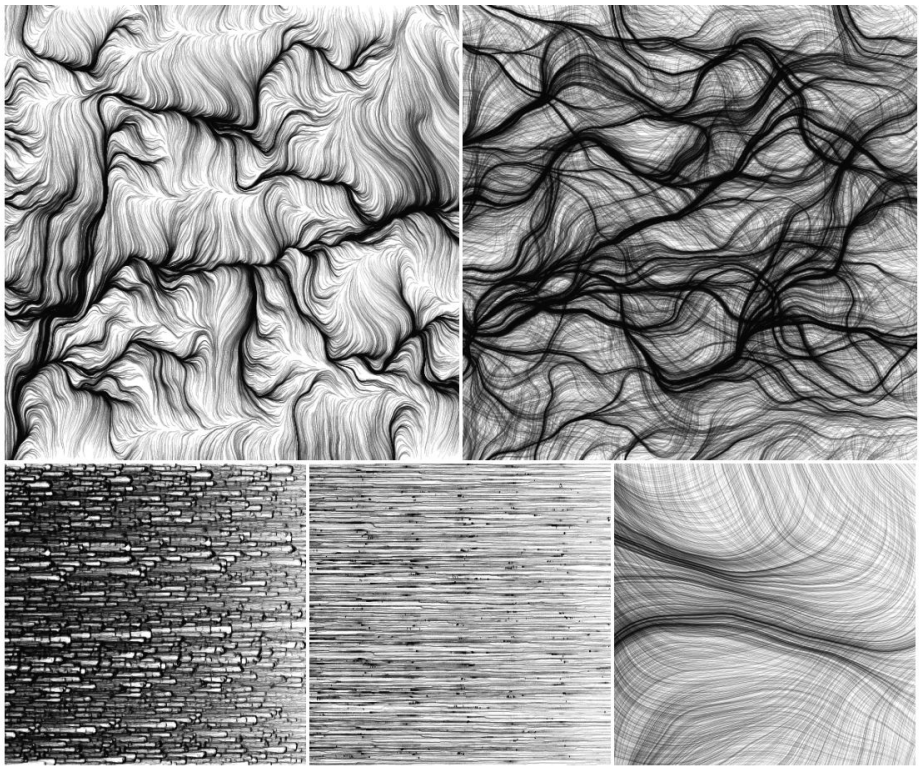
https://en.wikipedia.org/wiki/Centripetal_Catmull%E2%80%93Rom_spline

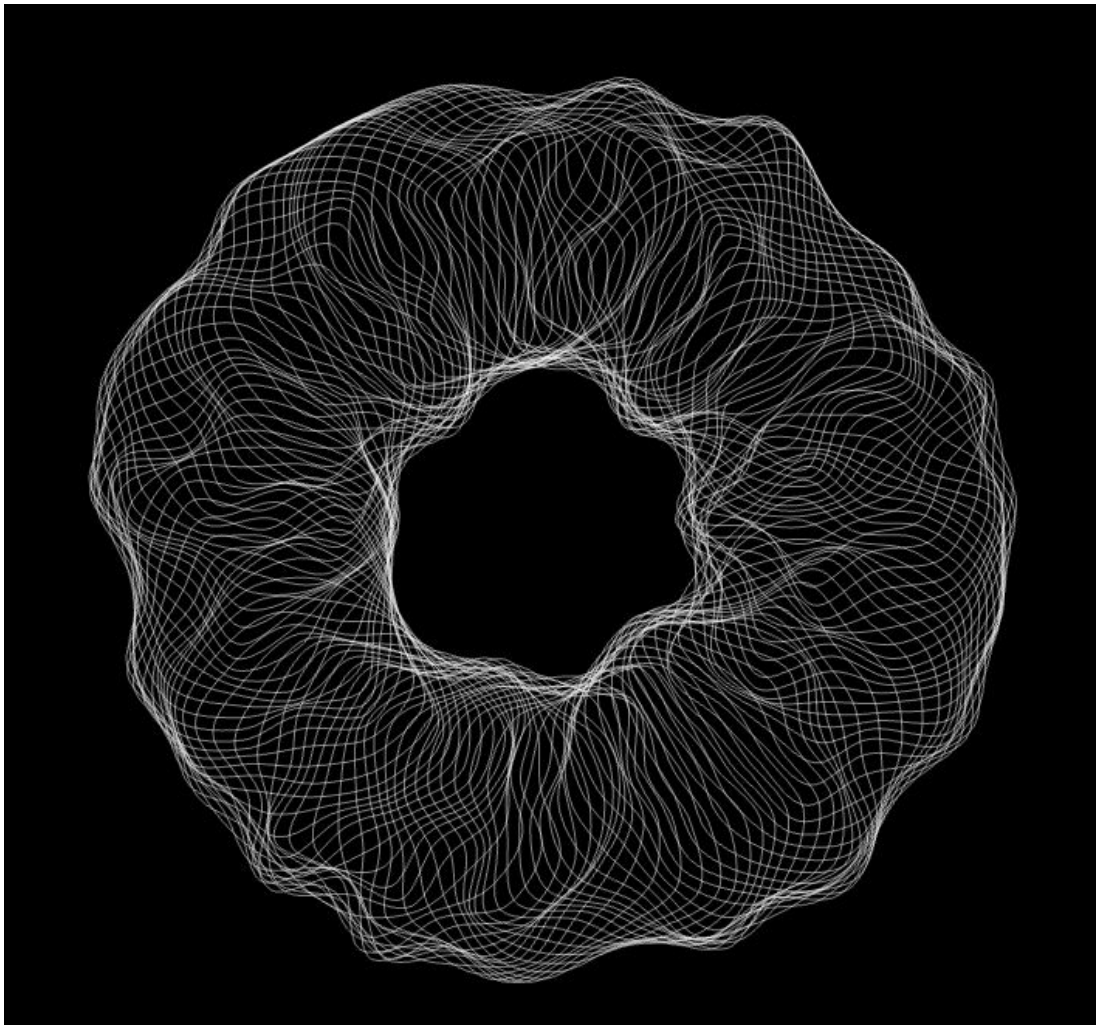
Each line one breath

John Franzen

https://www.johnfranzen.com/each-line-one-breath

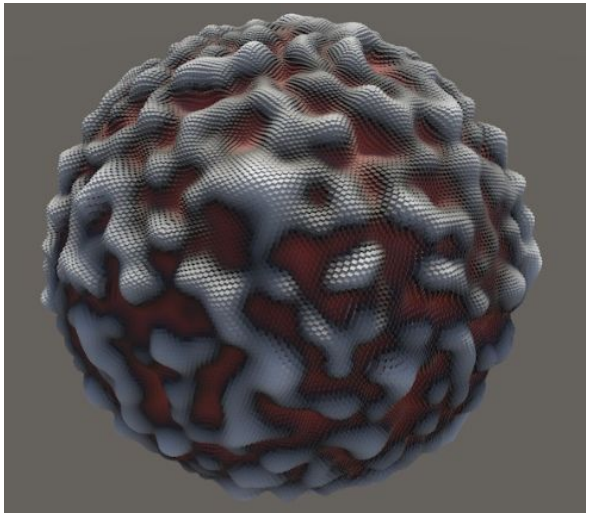
But what if we want to do stuff like this?



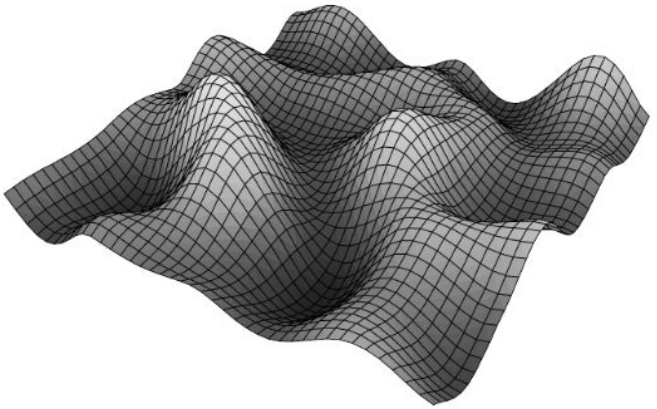


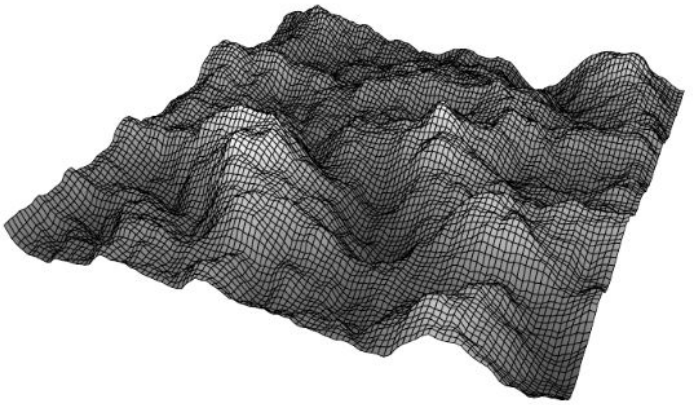
https://flashyprogramming.wordpress.com/2009/12/26/visualizing-perlin-noise/

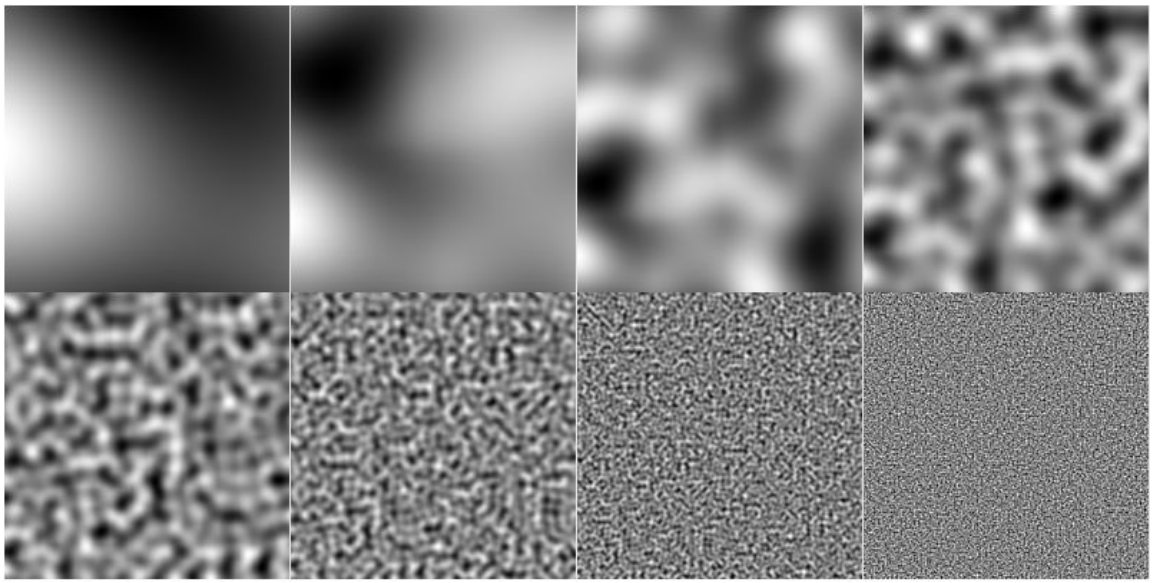
Enter Perlin Noise



- invented by Ken Perlin
  - Computer graphics pioneer
- A bit like “pepper” of computer graphics
  - Very useful
  - Don’t use too much:)
- Very useful to create organic forms







https://cs.nyu.edu/~perlin/

Intermezzo - **Demo Scene**
Elevated by Inigo Quilez

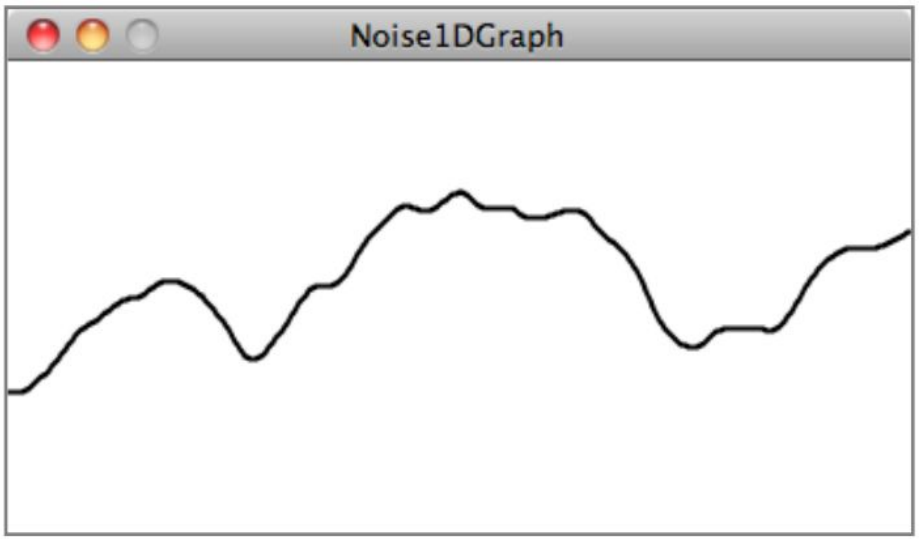
Video: https://www.youtube.com/watch?v=jB0vBmiTr6o
ShaderToy: https://www.shadertoy.com/view/MdX3Rr
https://www.youtube.com/channel/UCdmAhiG8HQDlz8uyekw4ENw

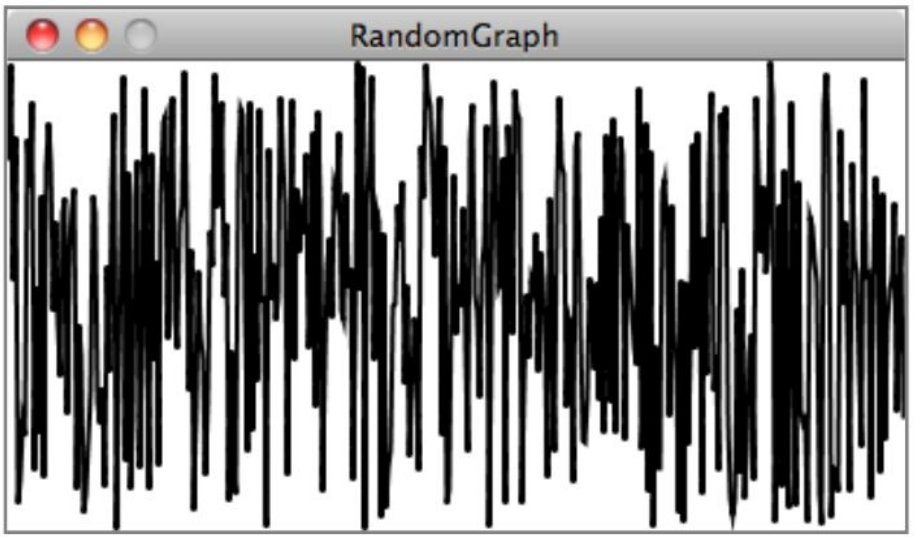
https://www.shadertoy.com/view/4ttSWf

`ofNoise()`

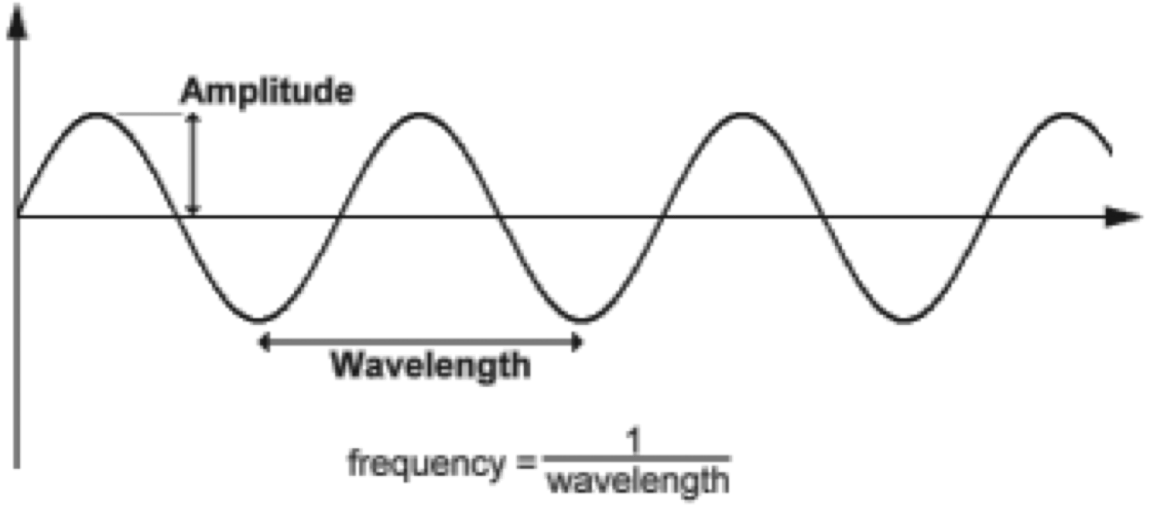
the details

- `ofNoise()` returns a value between 0-1
- **`ofSignedNoise()`** returns a value between -1 and 1
- we pass as parameters the point in time we want
- ie `ofNoise(5)` always returns the same value

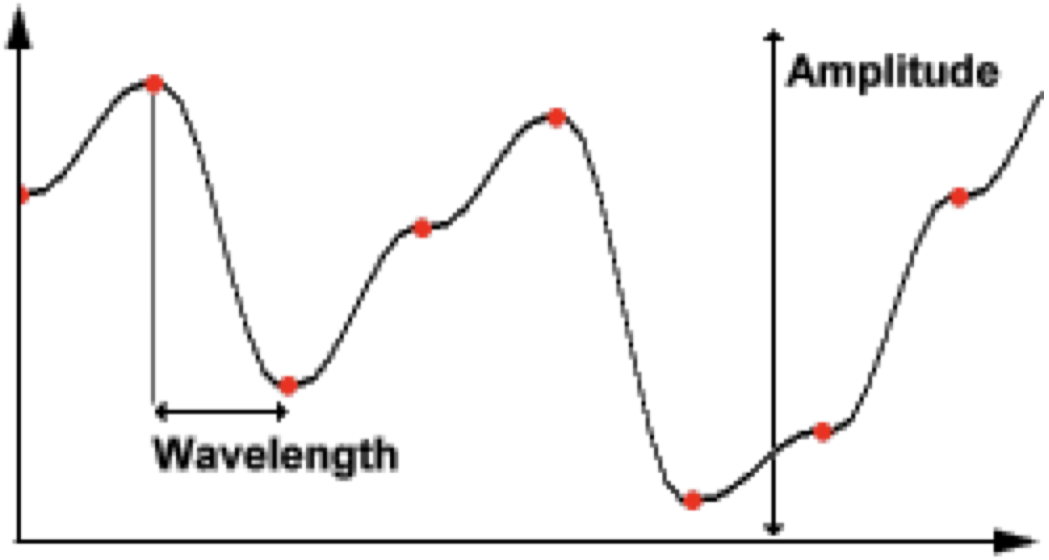




Noise waves

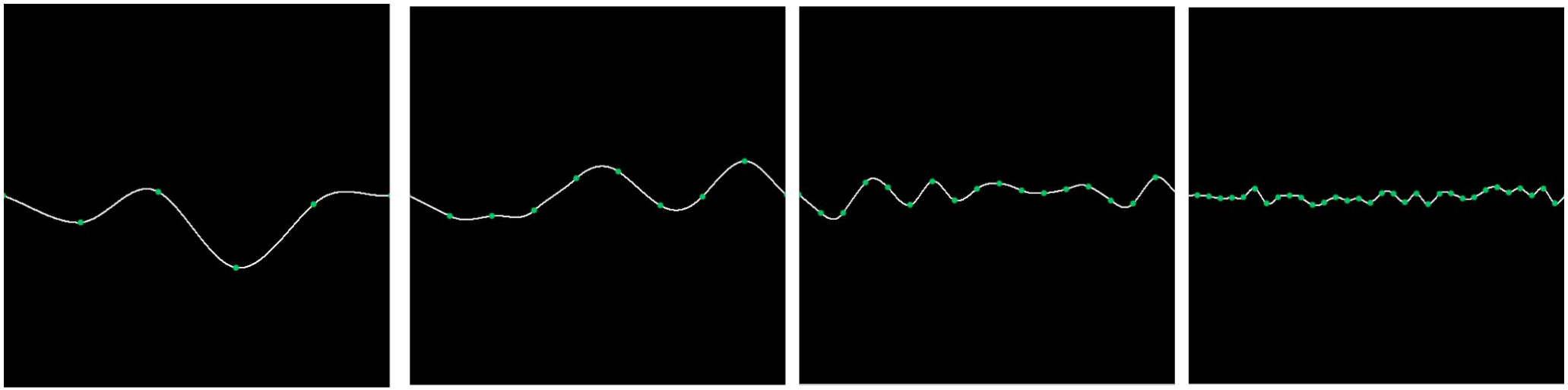


Sine wave



Noise wave

Controlling noise waves



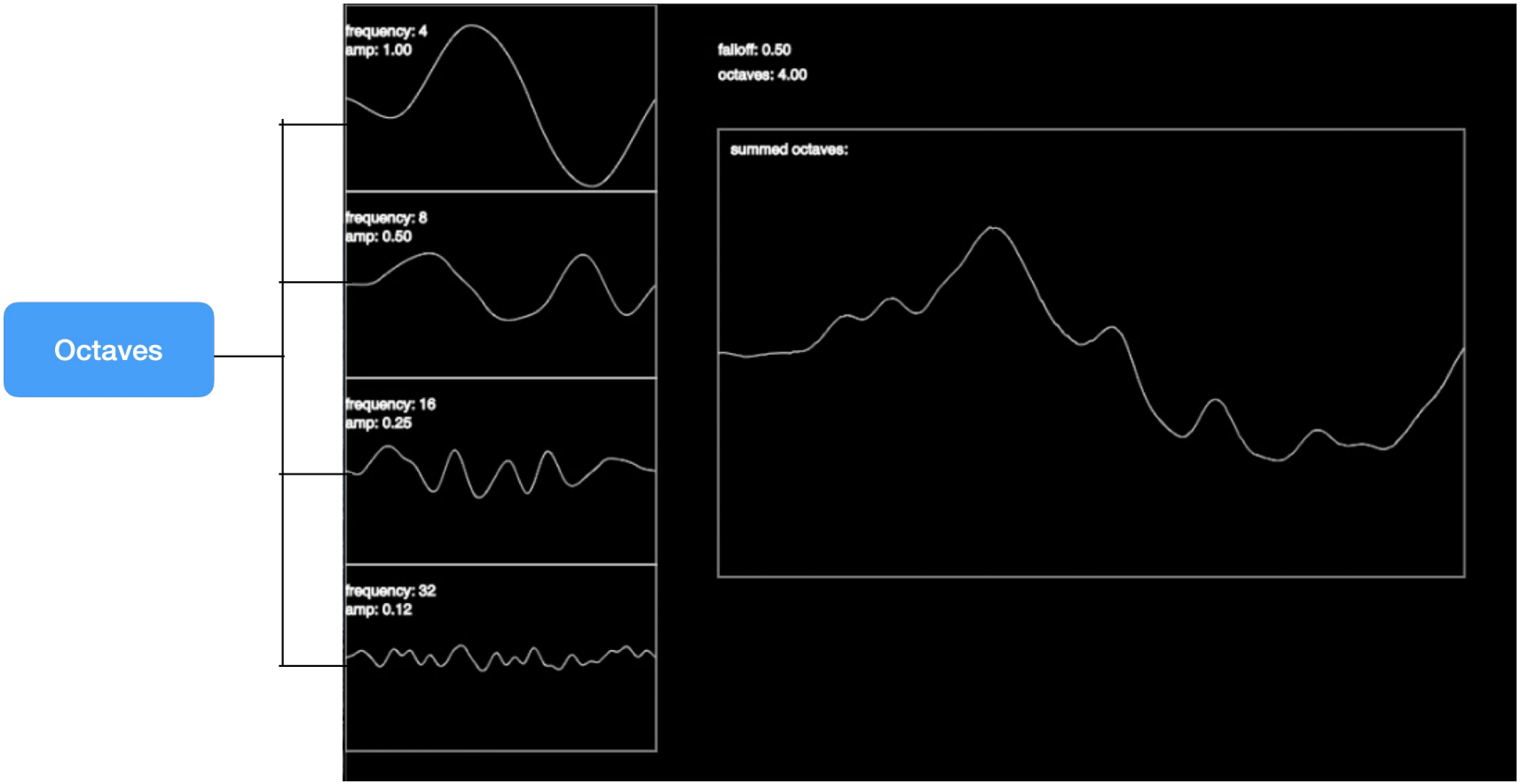
freq = 4
amp = 1

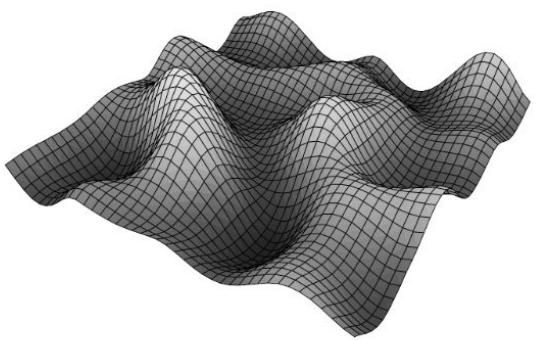
freq = 8
amp = 0.5

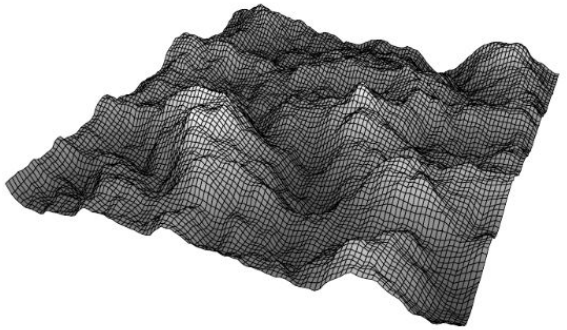
freq = 16
amp = 0.25

freq = 32
amp = 0.125

Adding noise waves (octave noise)





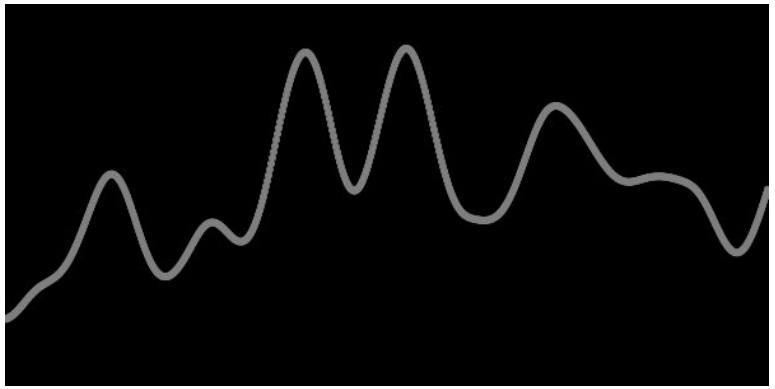


octaveNoise

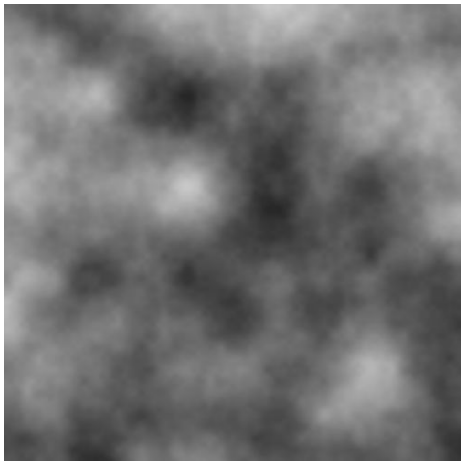
`ofNoise()`

the details

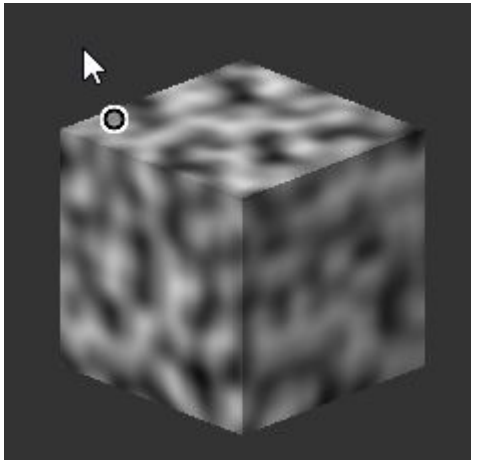
- `ofNoise()` always returns one float
- But **input** can be either 1 value, 2 values or 3 values
  - Imagine these as a randomized:
    - Waveform (1d)
    - Image (2d)
    - Volume (3d)



1d A moving slice of ->



2d A moving slice of ->

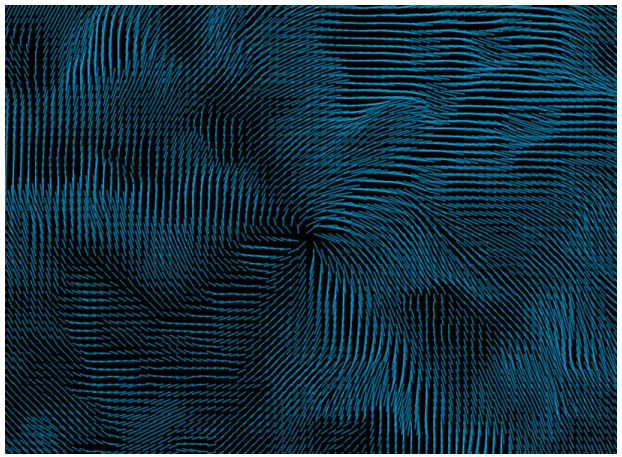


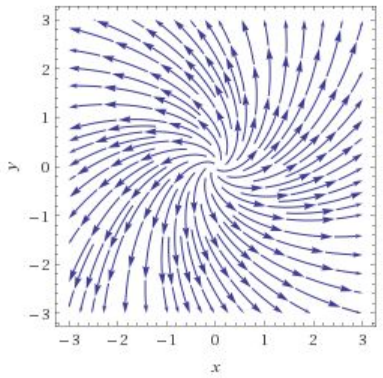
3d

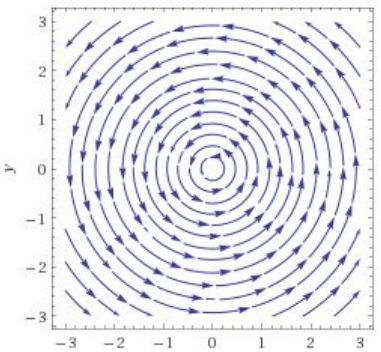
`octaveNoise`

2d example: Vector fields

- We can use 2d (or a slice of 3d) noise to compute a direction for a given XY position (as an angle)
  - `float angle = ofSignedNoise(pos.x, pos.y)*PI;`
- Use direction to compute a “flow” vector
  - `vec2 flow = vec2(cos(angle), sin(angle))`
    - A unit vector (length 1)
- The result is a vector field (a vector for any given position)
- We can animate it (e.g by moving in 3d)
  - `float angle = ofSignedNoise(pos.x, pos.y, z)*PI;`
- We can add vector fields together
  - Just add the vectors resulting for a given position

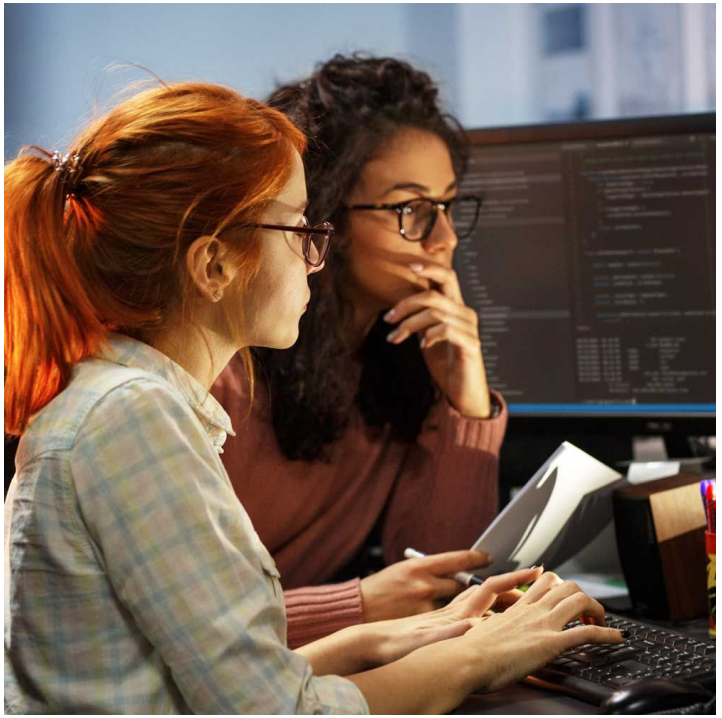


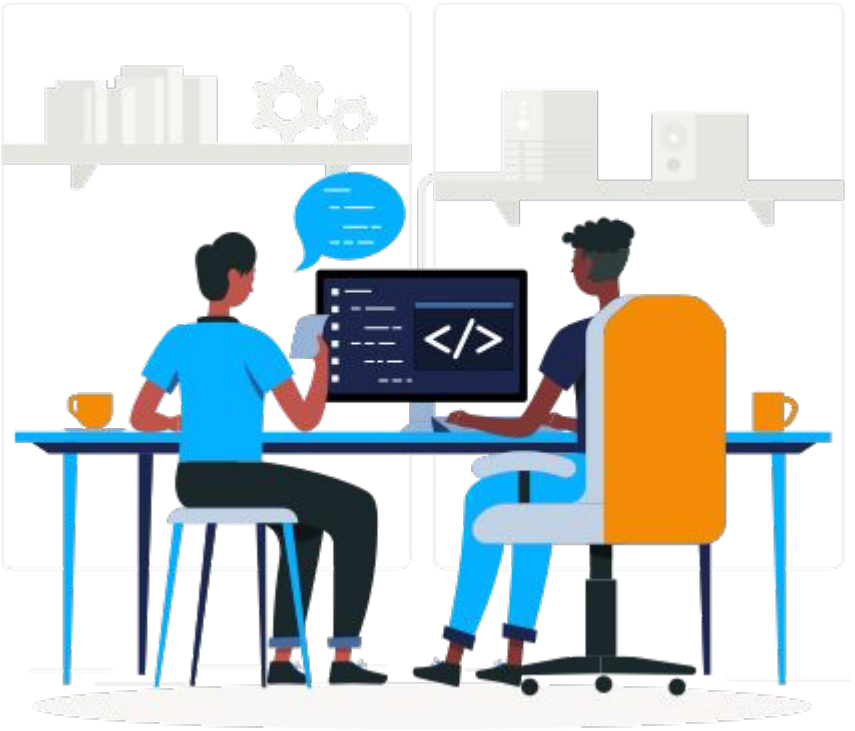




https://en.wikipedia.org/wiki/Vector_field

Pair programming





- **Driver:** types code // reads instructions
- **Navigator:** reads instructions // looks up commands // corrects code