programming for artists and designers

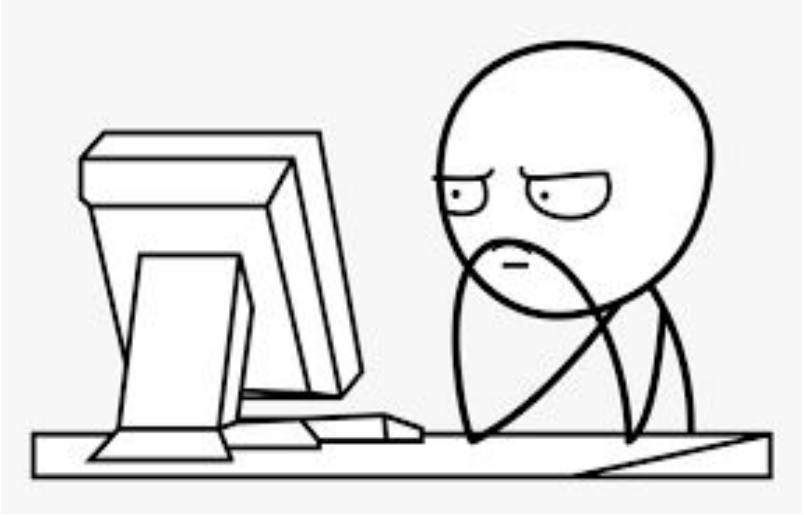
Daniel Berio

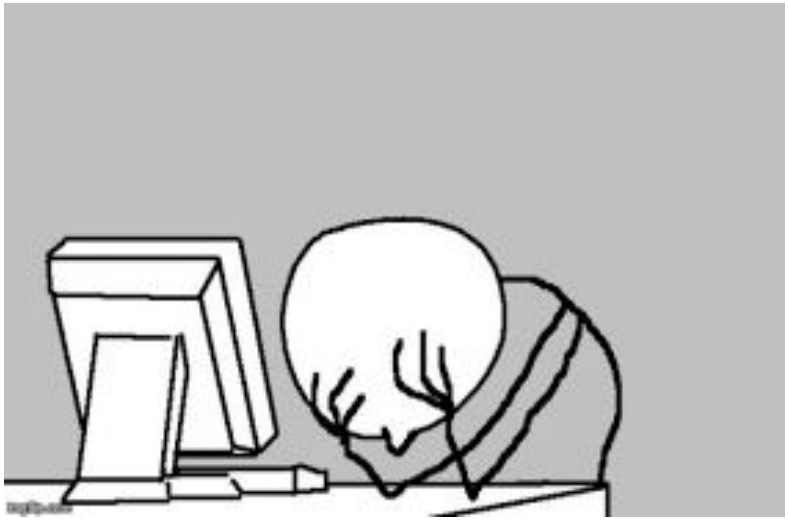
Session Summary

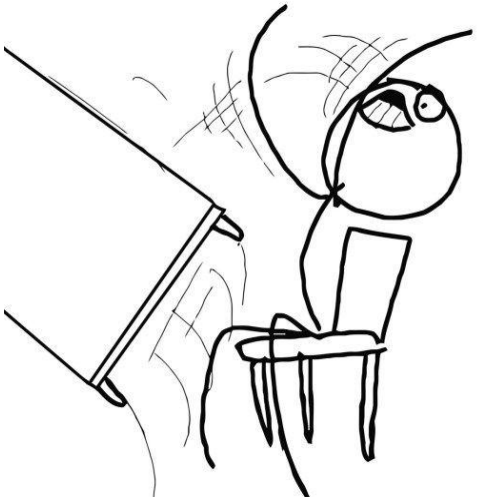
- Modularity: Function review
- C++ data structures (high level)
- Dynamic arrays (std::vector)
- Modularity and transformation review:
  - Turtle graphics
- Lab + Mid-term project work time.
  - *We might do some of the review in groups as part of the project work time.*

Learning to Code

this is still me today



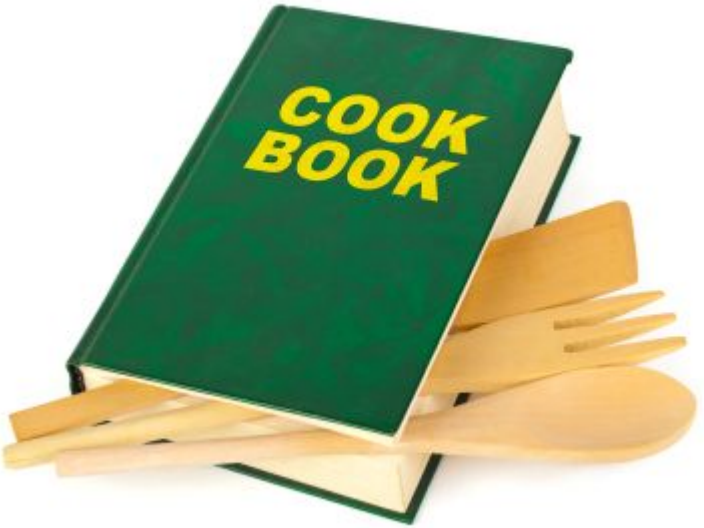




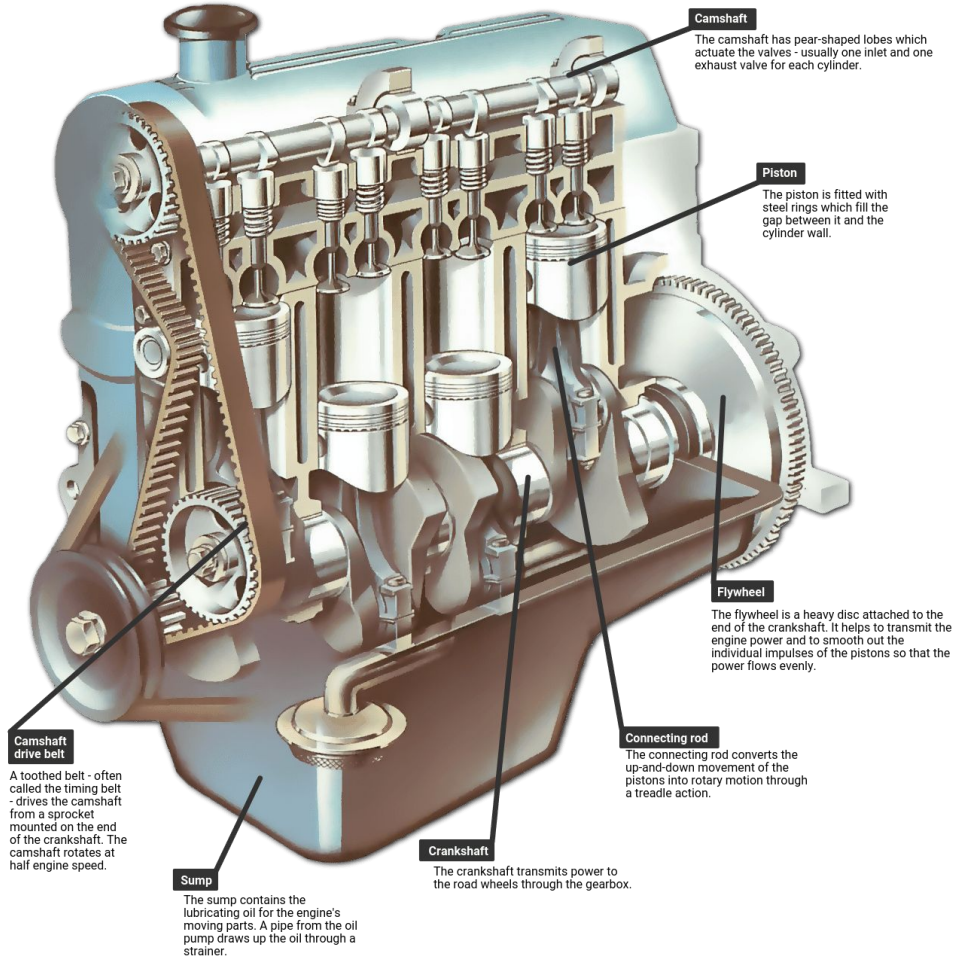
“I mostly understand, I’m just not sure how to do things on my own”

coding vs cooking

- we’ve seen lots of recipes already
- cookbooks are important for learning
- eventually you get enough confidence to mix up a few recipes to make something new
- eventually great chefs don’t need recipes



Where we are now: study the recipes, maybe try mix a few together



✗ BUILD A VEHICLE??

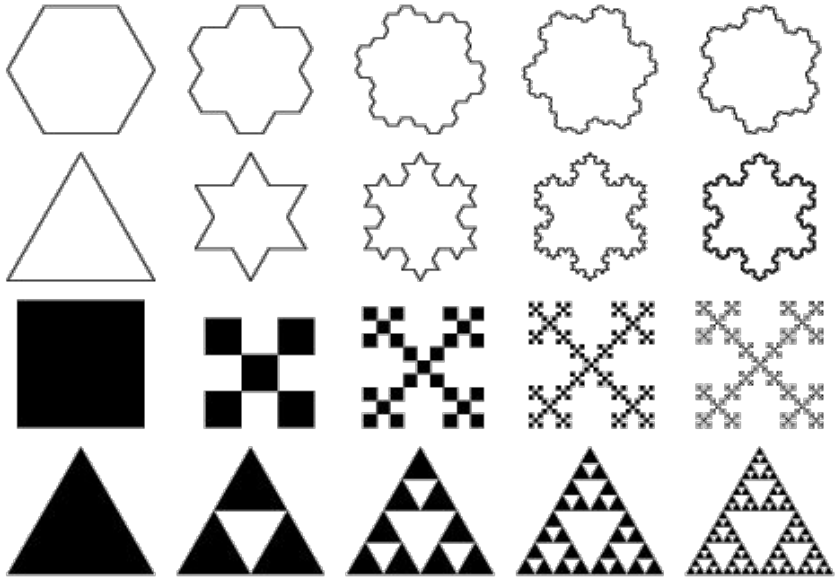
✓ Here is an engine:

- Study it
- Break it
- Fix it
- Make a new thing from the pieces

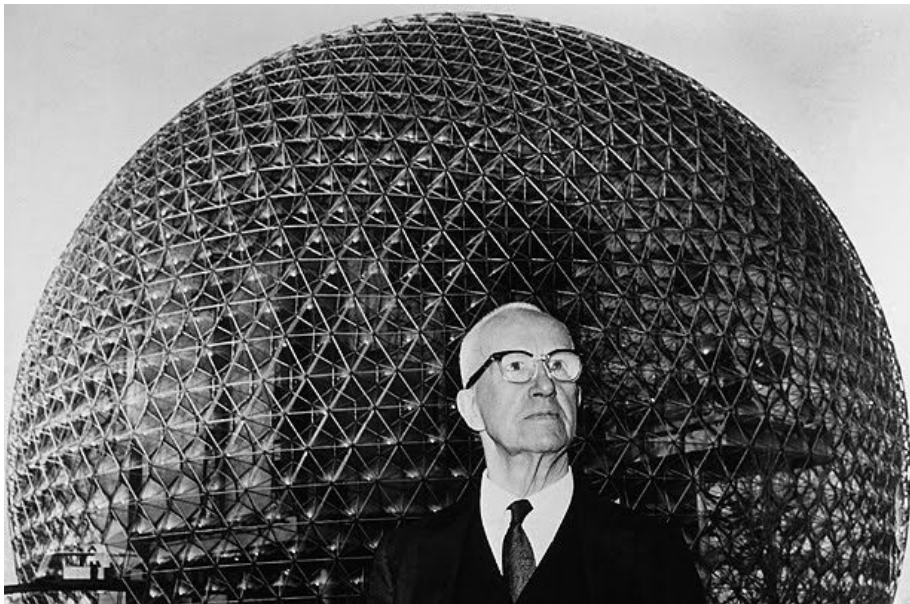

NAPIER
OIL
IGN
BOOST

Modularity

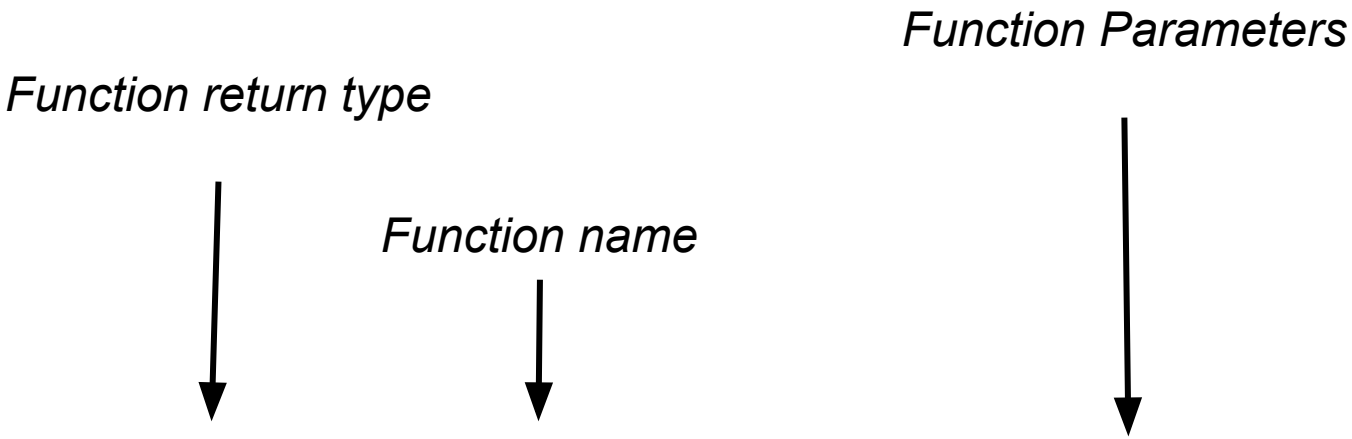
- in typography
  - typefaces are very modular
- in software it is used for optimization
  - produce complex images from small group of forms
    - 90's site wallpapers
    - fractals
- in construction
  - buildings are made by standardized elements
  - Buckminster Fuller took the idea to the extreme



abcdefghijklm
nopqrstuvwxyz

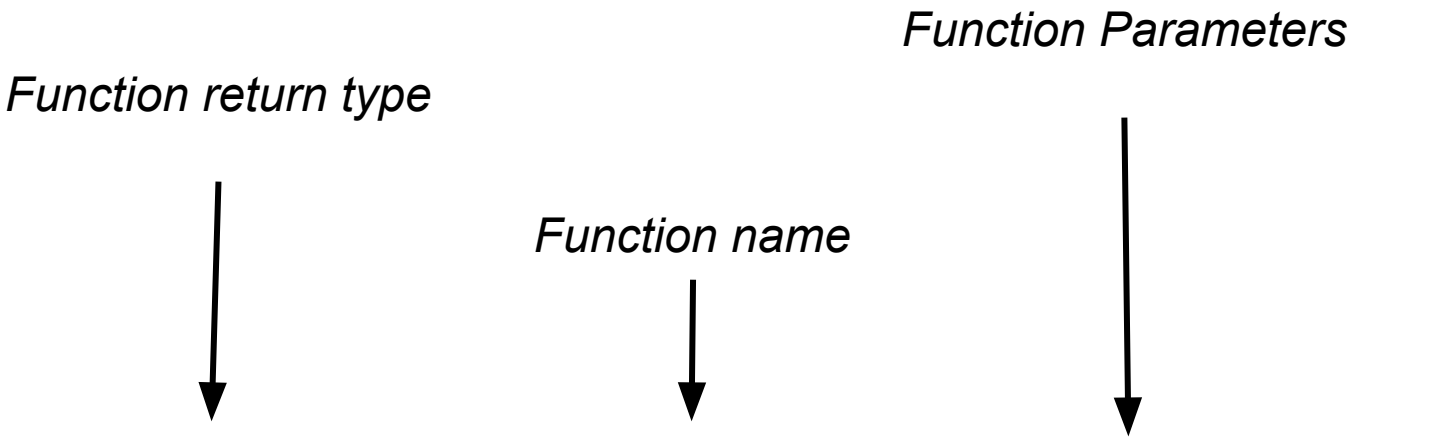


Anatomy of a (global) function




The void!

Anatomy of a class function



Anatomy of a class function

Defining it in ofApp.h (inside the scope of `class ofApp`):

```
float multiply(float x, float y);
```

In ofApp.cpp:

```
float ofApp::multiply(float x, float y){
   float multiplied = x * y;
   return multiplied;
}
```

Using it in update in ofApp.cpp:

```
float myNumber = multiply(100,20);
```

Modularity

- Modularity helps us make code more reusable
- We don’t have to repeat ourselves as much
- Helps us keep our code neat

For example with a nested for loop:

```
for (int x = 0; x < 10; x++)
{
    for (int y = 0; y < 10; y++)
    {
        // call a function
    }
}
```

Modularity

- Modularity helps us make code more reusable
- We don't have to repeat ourselves as much
- Helps us keep our code neat

For example with a nested for loop:

```
for (int x = 0; x < 10; x++)
{
    for (int y = 0; y < 10; y++)
    {
        // call a function
        // call another function
    }
}
```

Modularity

- Modularity helps us make code more reusable
- We don't have to repeat ourselves as much
- Helps us keep our code neat

For example with a nested for loop:

```
for (int x = 0; x < 10; x++)
{
    for (int y = 0; y < 10; y++)
    {
        // call a function
        // call another function
    }
}
```

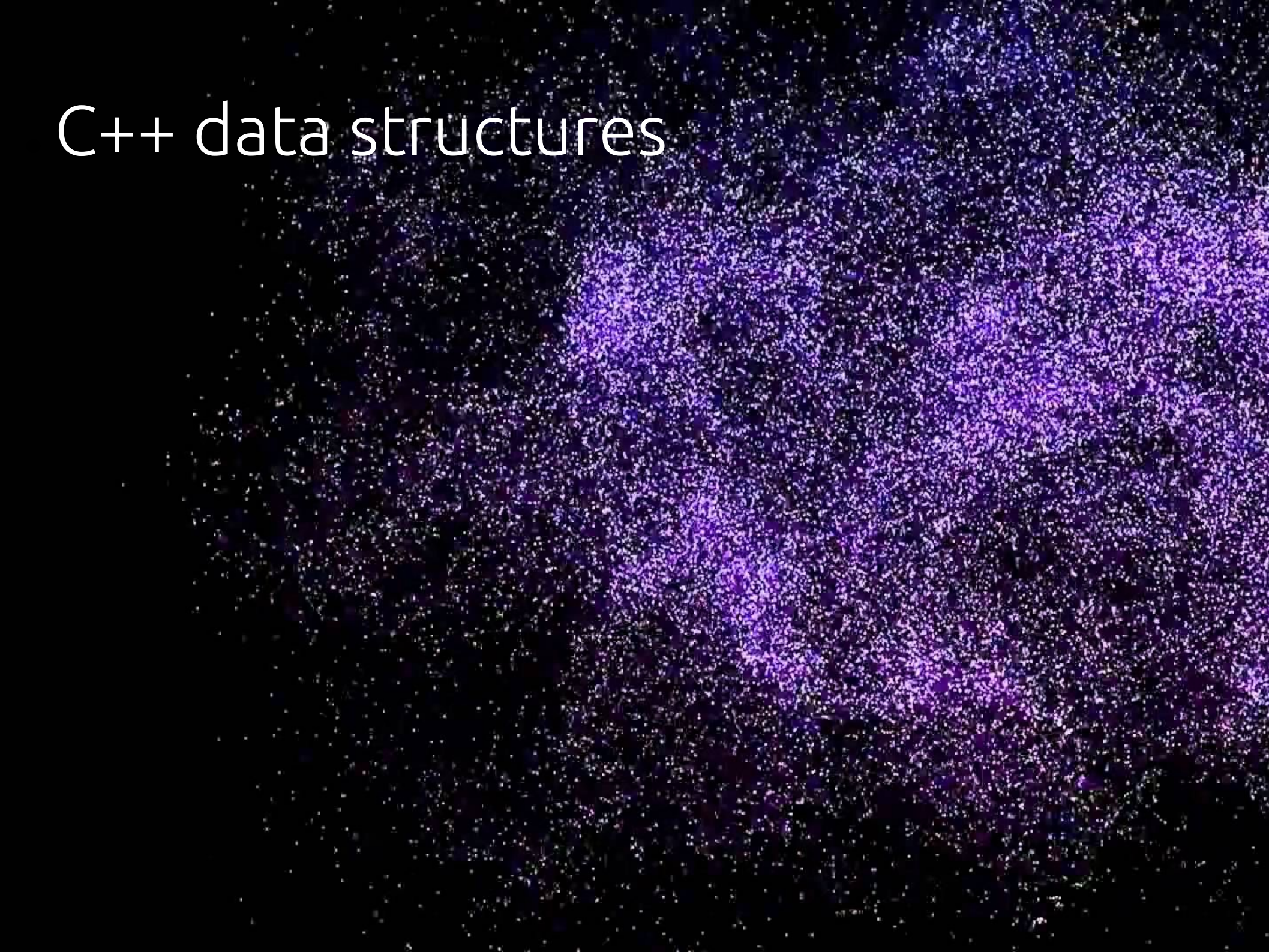
Modularity

- Modularity helps us make code more reusable
- We don’t have to repeat ourselves as much
- Helps us keep our code neat

For example with a nested for loop:

```
for (int x = 0; x < 10; x++)
{
    for (int y = 0; y < 10; y++)
    {
        // call a function
        // change function
    }
}
```

C++ data structures



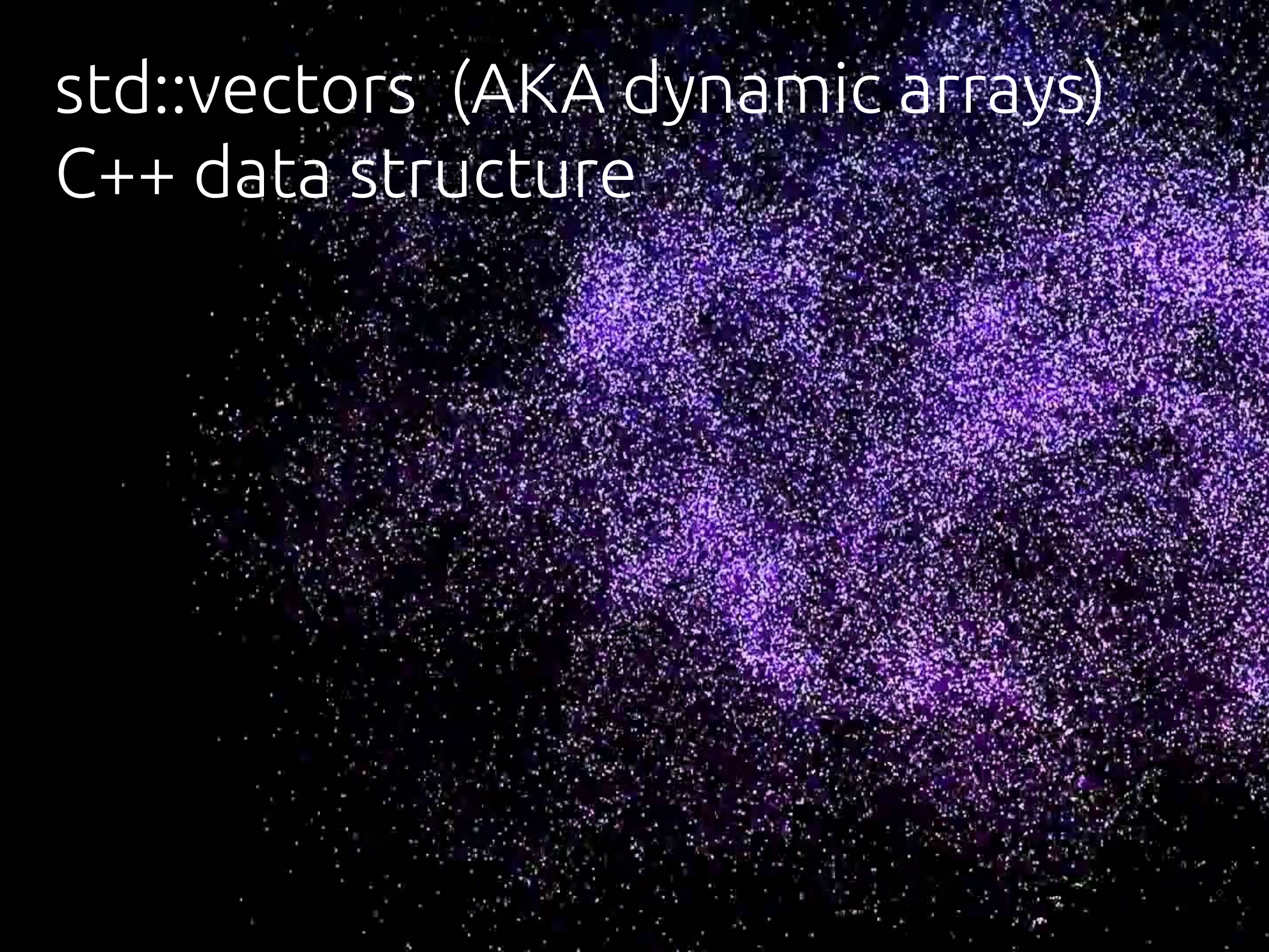
C++ data structures

- Special (structured) types, we have seen a few already
  - `vec2, ofColor, string, …`
  - `ofApp` (we define it!)
    - Our app is a data structure, in ofApp.h and ofApp.cpp
- Defined in a library or by you (later on…)
  - Usually as a “class definition”
- Define a **type** but with additional features

C++ data structures

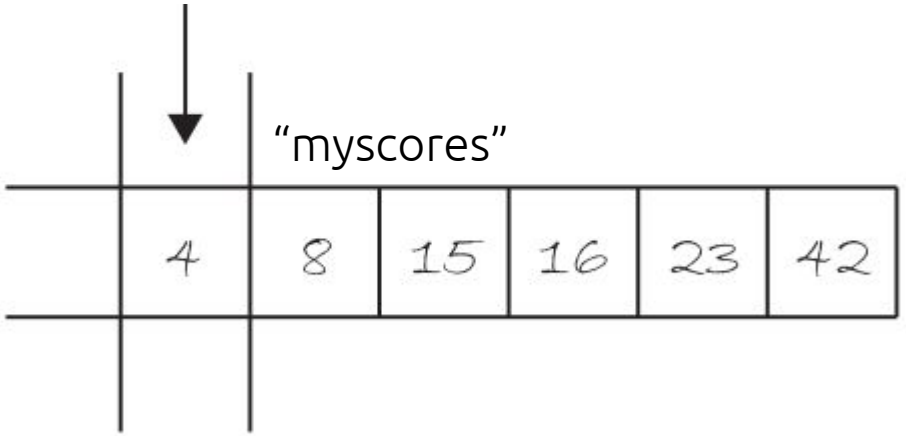
- Define a **type** but with additional features:
  - A data structure can have properties
    - Essentially **variables** that are specific to the data structure
    - E.g. with `vec2` we can get and set the `.x` and `.y` properties
    - Usually accessed with “dot notation”, e.g.
      - `myVec.x = 10;`
  - A data structure can have “methods”
    - Essentially **functions** that are specific to the data structure
    - E.g. with `string` we can get the length with `.length()`
    - Again with “**dot notation**”, e.g.
      - `int len = myString.length();`
  - We create a data structure with a “constructor”
    - Similar to a function, but using the structure name
    - Can have parameters, e.g.
      - `vec2 myVec = vec2(10, 20);`

std::vectors (AKA dynamic arrays)
C++ data structure

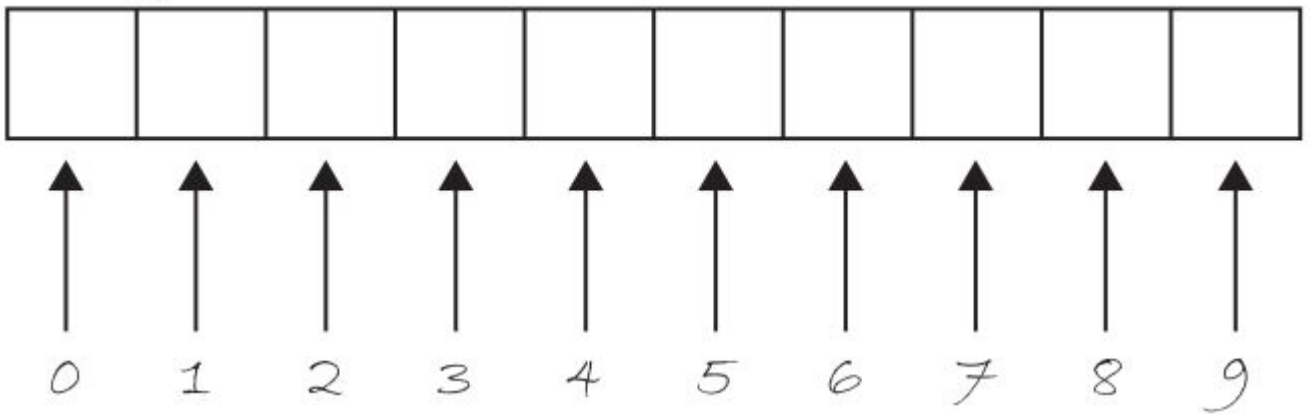


std::vector

value at position 0



index of values



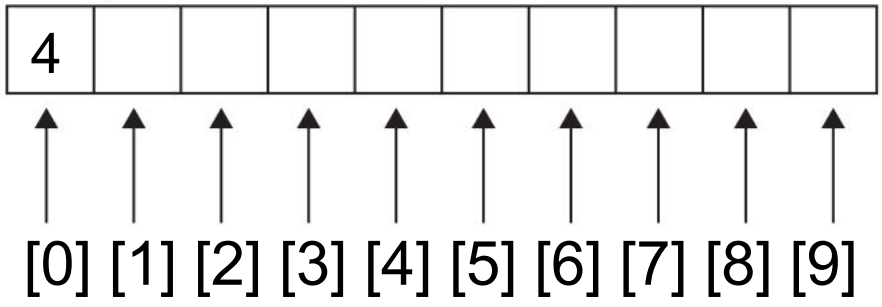
How are vectors different from arrays?

Arrays have a fixed size

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) !!!!!!

How do we use a vector?

Unlike arrays in C++, we can *dynamically* add or remove things from C++ vector data structures

Notice we need to say what type we want to store

```
vector<int> myScores;                    // vector declaration
myScores.push_back(80);                  // adding value at the end of vector
myScores[0];                             // read the first value
myScores.back();                         // read the last value
cout << myScores.size() << endl; // getting vector size
myScores.pop_back();                     // removing last value from end
```

...other examples

```
vector<int> myGrades;
vector<float> myTemperatures;
vector<vec2> myPositions;
```

Question: What do we do if we have a vector with 1000 elements?

```
vector <float> circleX;
circleX.push_back(ofRandom(0,200));
circleX.push_back(ofRandom(0,200));
circleX.push_back(ofRandom(0,200));
…
circleX.push_back(ofRandom(0,200));
```

Answer

We can use a for loop!!!!

```
vector<float> circleX;

for (int i = 0; i<10000; i++)
{
  circleX.push_back(ofRandom(0,200));
}
```

Modularity:
A function can return a vector

E.g. we might want to define a function returning the positions in a grid:

```
vector<vec2> gridPoints(int numRows, int numCols)
```

Then in ofApp::draw()...

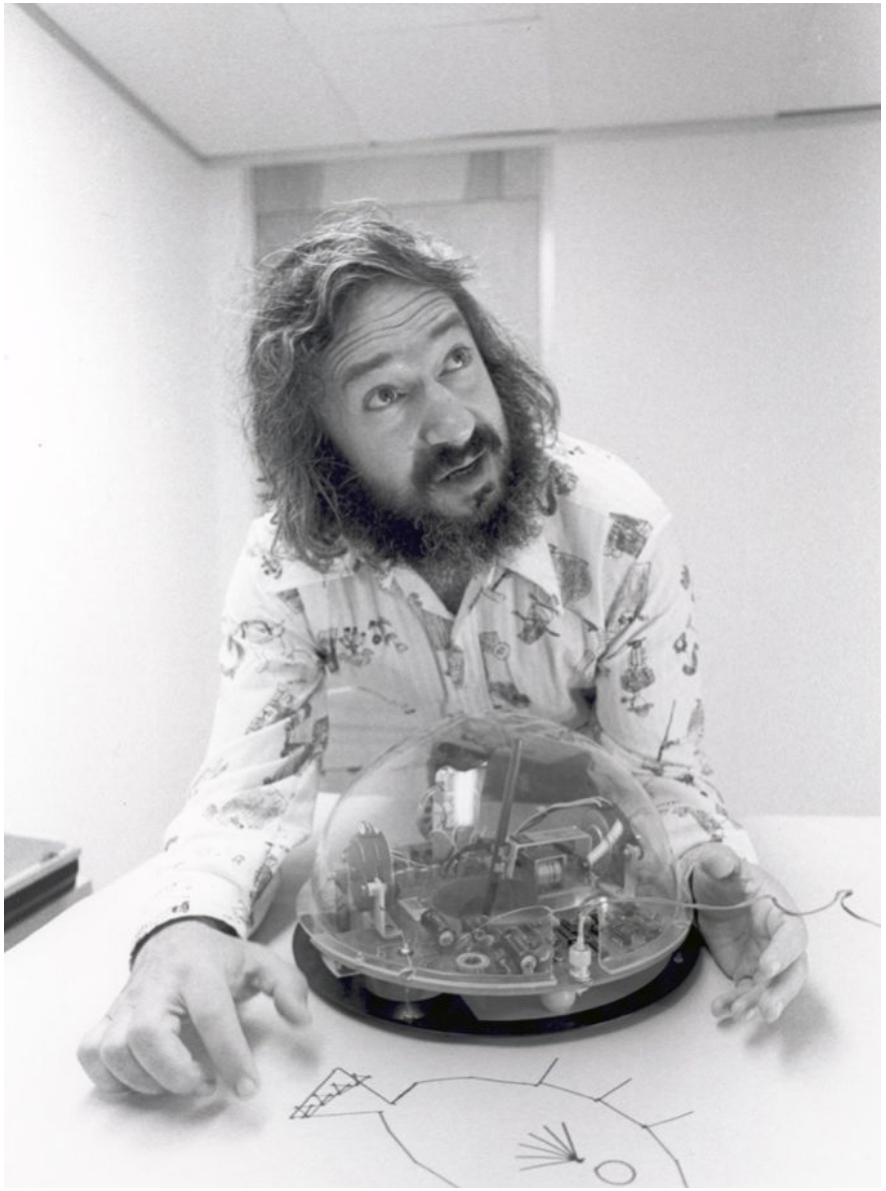
```
vector<vec2> points = gridPoints(10, 20);

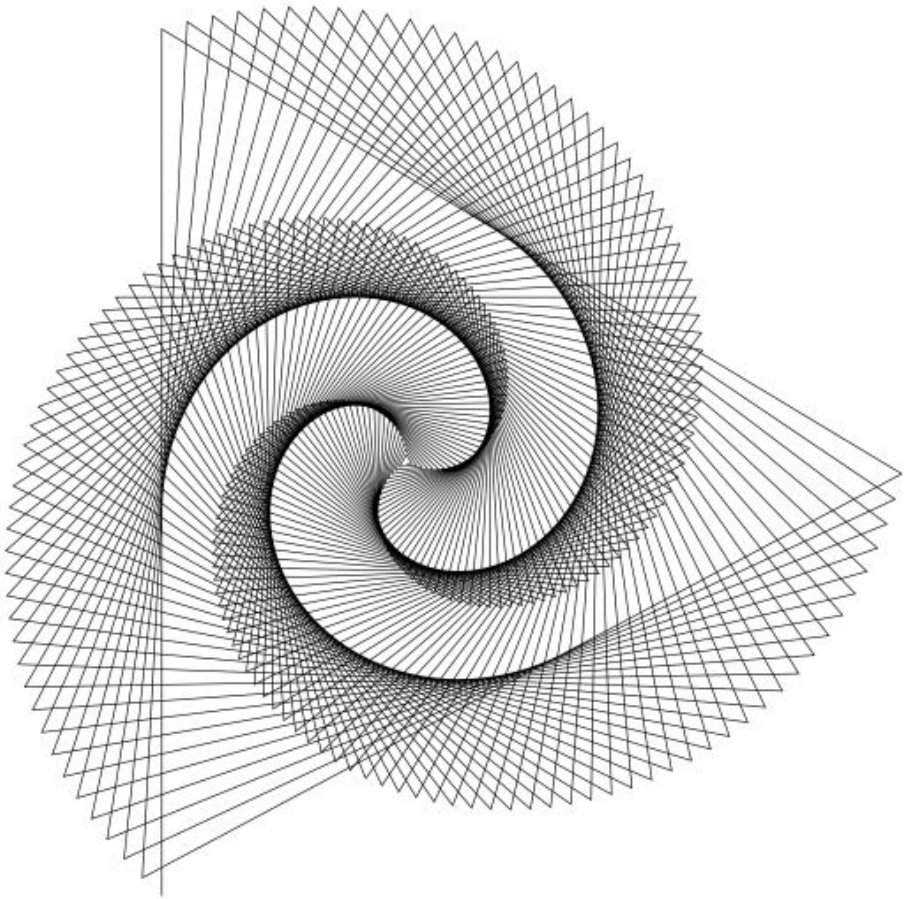
// draw all points in grid
ofSetColor(255);
for (int i = 0; i < points.size(); i++) {
  ofDrawCircle(points[i], 5);
}
```

gridArray

Break

Modularity and transformations: **Turtle graphics**

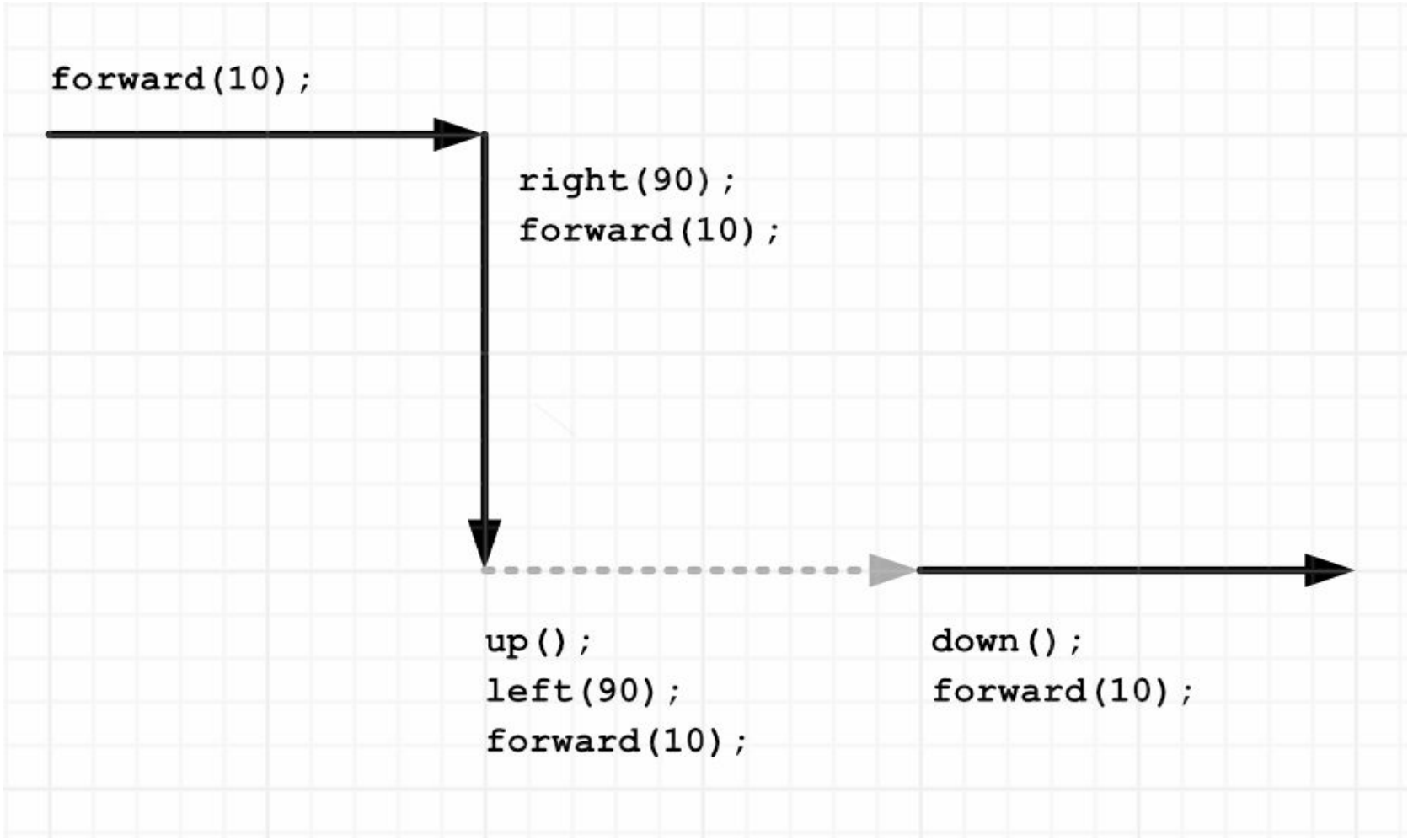




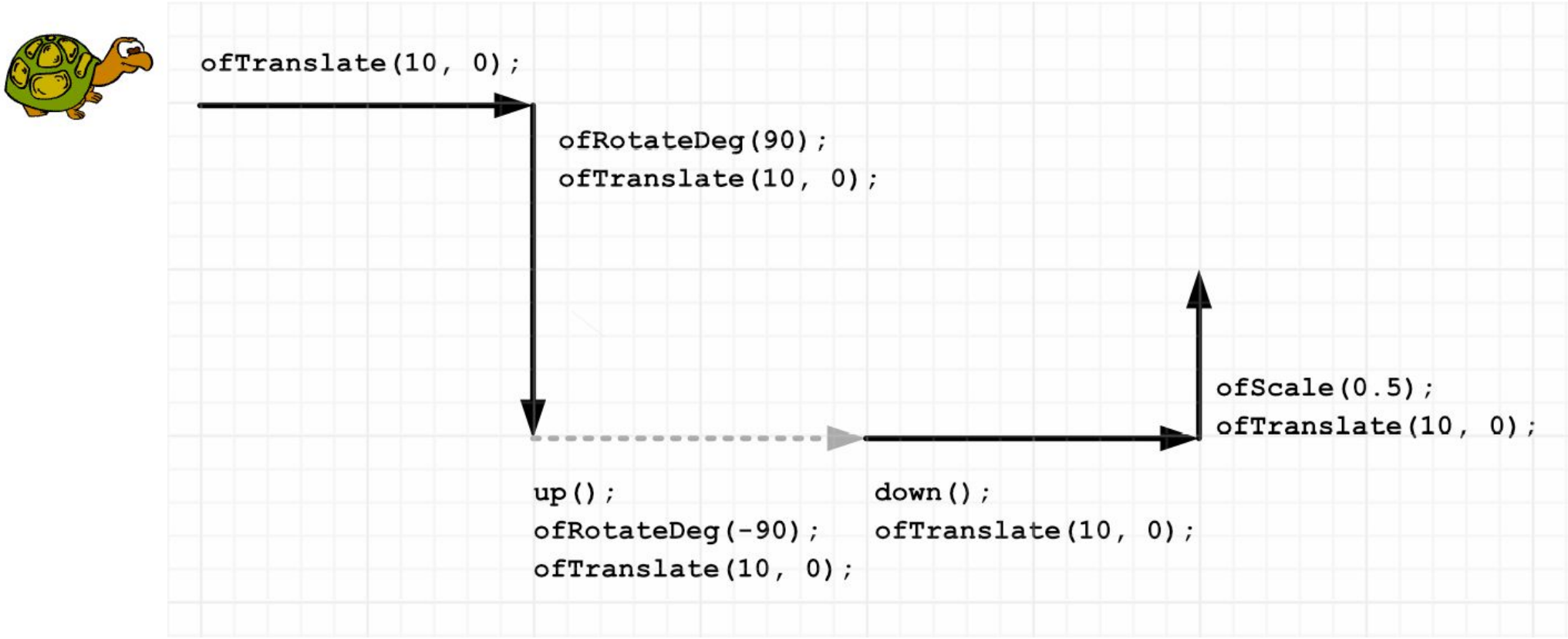
https://turtletoy.net/

https://el.media.mit.edu/logo-foundation/what_is_logo/logo_and_learning.html

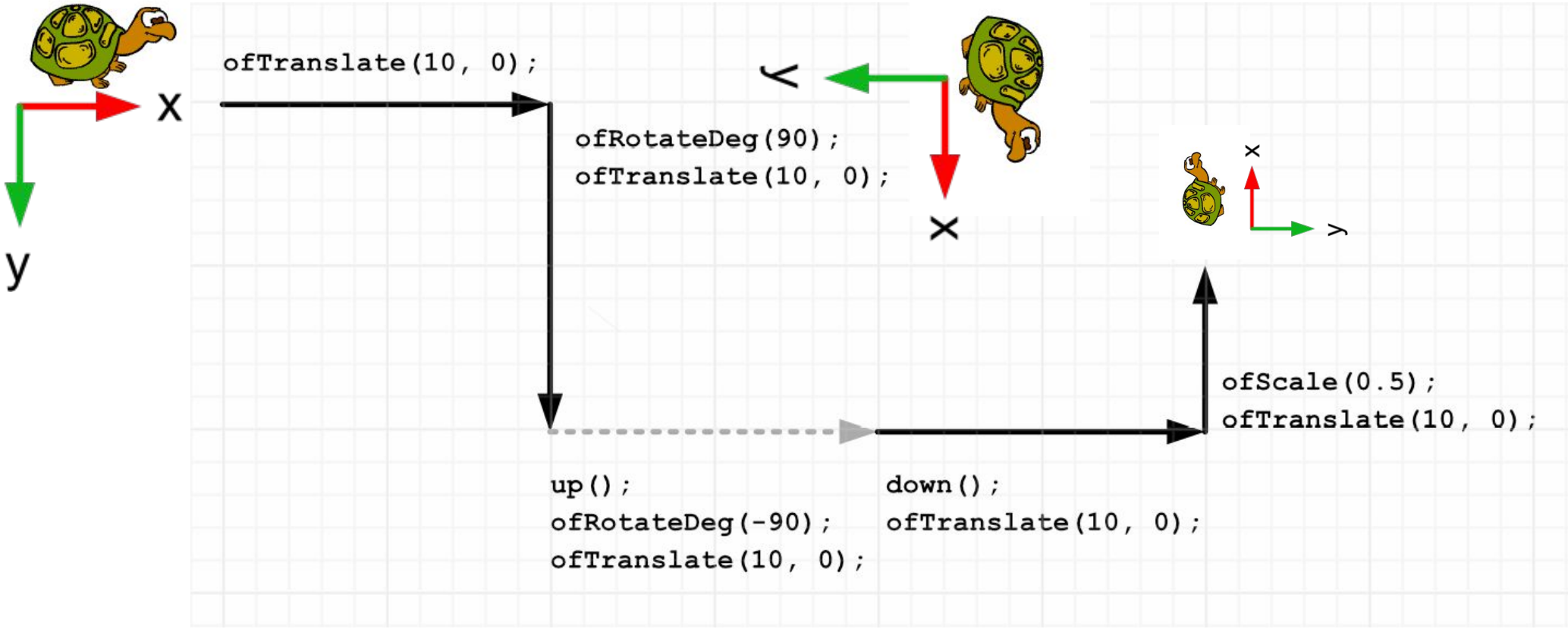
Modularity and transformations: **Turtle graphics**



Modularity and transformations: **Turtle graphics**



Modularity and transformations: **Turtle graphics**

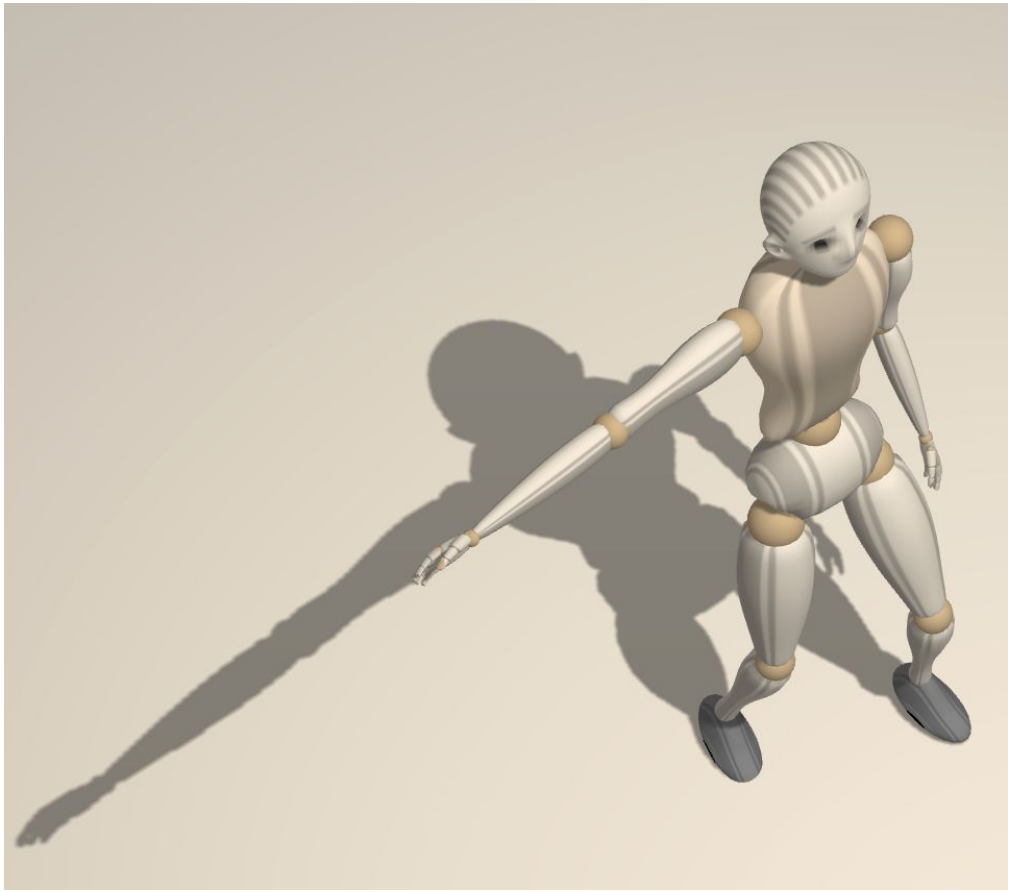


- Translation: The origin (0,0) always coincides with the turtle.
- Rotation: The x direction is the direction in which the turtle is facing.
- Scale: The distance travelled by the turtle is proportional to the turtle size.

Be a turtle

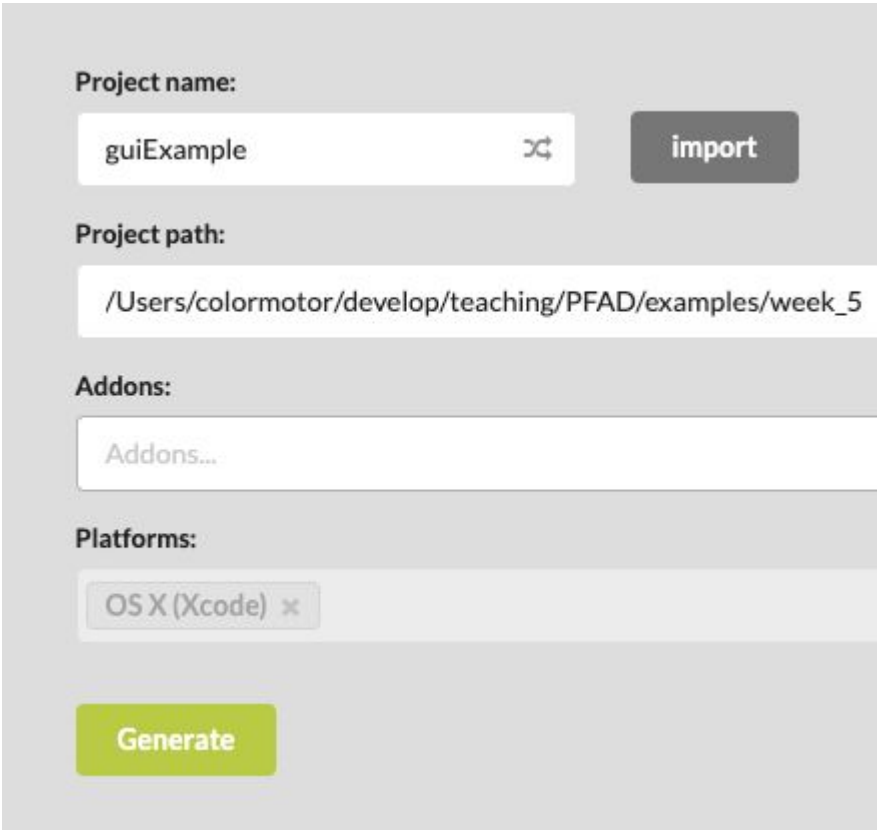
Forward (X axis)

Along right arm (Y axis)



turtleExample

Before the lab: ofxGui



- Allows you to add a UI
  - to control parameters
- Must import addon ofxGui
- Declare controls in ofApp.h
  - Inside class definition, e.g:
    - ofxPanel gui; // The gui
      ofxFloatSlider radius; // A control
- Initialize in ofApp.cpp
  - In `ofApp::setup()`, e.g:
    - gui.setup();
      gui.add(radius.setup("Radius", 100, 10, 250));
- Use like a variable of the corresponding type, e.g. for a float slider:
  - ofDrawCircle(100, 100, radius);
- Draw the GUI:
  - gui.draw();

guiExample

Export one “frame” of drawing as SVG

WON’T WORK WITH ofSetBackgroundAuto(false)

In ofApp.h (inside class definition) or in ofApp.cpp (global):

```
bool exportVectorGraphics=false;
```

In draw():

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

In keyPressed() (if key==aKeyOfYourChoice, e.g. ‘ ‘ or ‘OF_KEY_RETURN’):

```
exportVectorGraphics = true;
```

If you do want to use persistent drawing

You will need “FBO”s to export high-res images

```
void ofApp::update(){
    canvasFbo.begin();
    ofPushMatrix();
    ofScale(fboScale);

    // Drawing code starts here
    ofTranslate(mouseX, mouseY);
    ofSetColor(0);
    ofDrawCircle(0,0,10);
    // Drawing code ends here

    ofPopMatrix();
    canvasFbo.end();
}
```

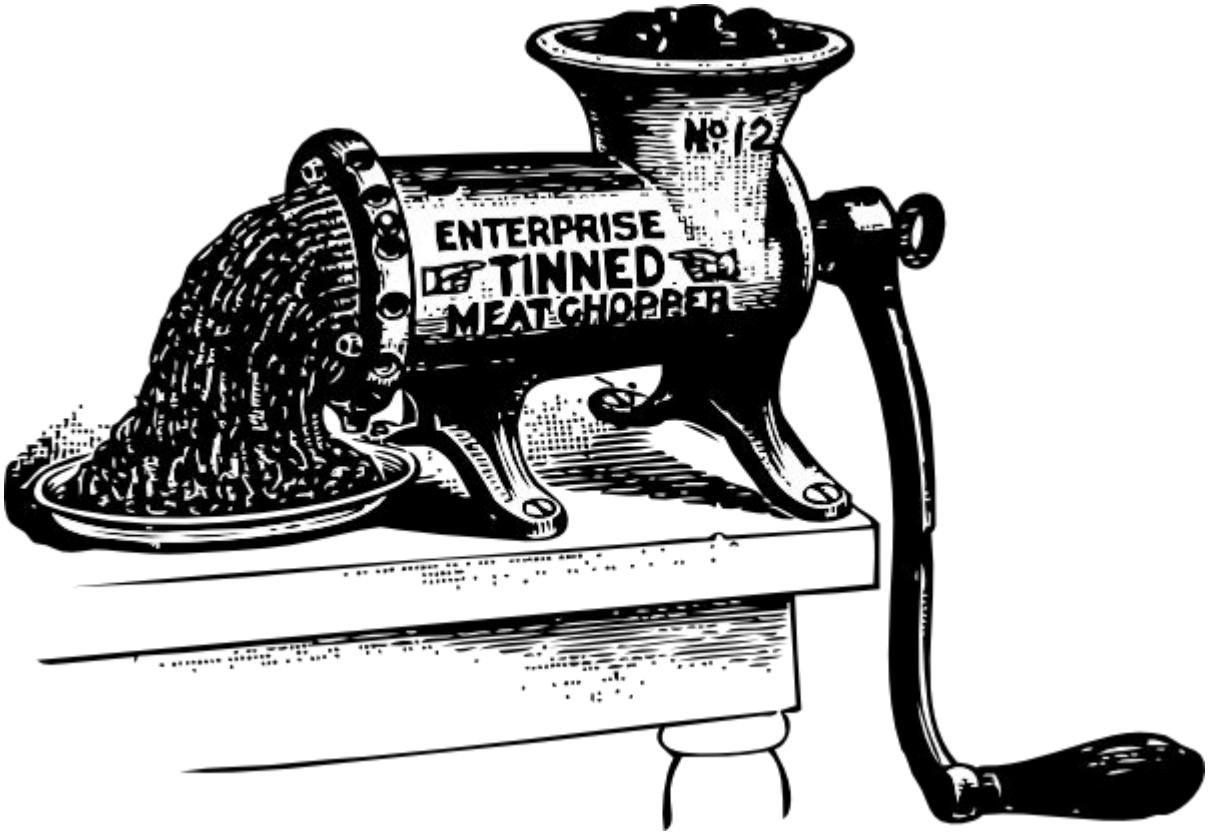
extraContent_FBOforPrint

Mid-term project work-time

- In **groups of three** (not your friends), for **5 mins each person** explains / shows idea & others ask questions & give suggestions (total 15 mins)
- **Focus groups** working with the same code, **30 mins.** Work together with others who are using the same starter code as you (e.g. the texture lab activity) or using similar approaches / methods, e.g. working with noise.
- Free work time for the rest of the session, work on your own code, ask questions.
- Set up working groups to meet outside class!

Review Slides: Number generators for Generative Art

(if we need them, look at last week's slides & extra recording for full detail)



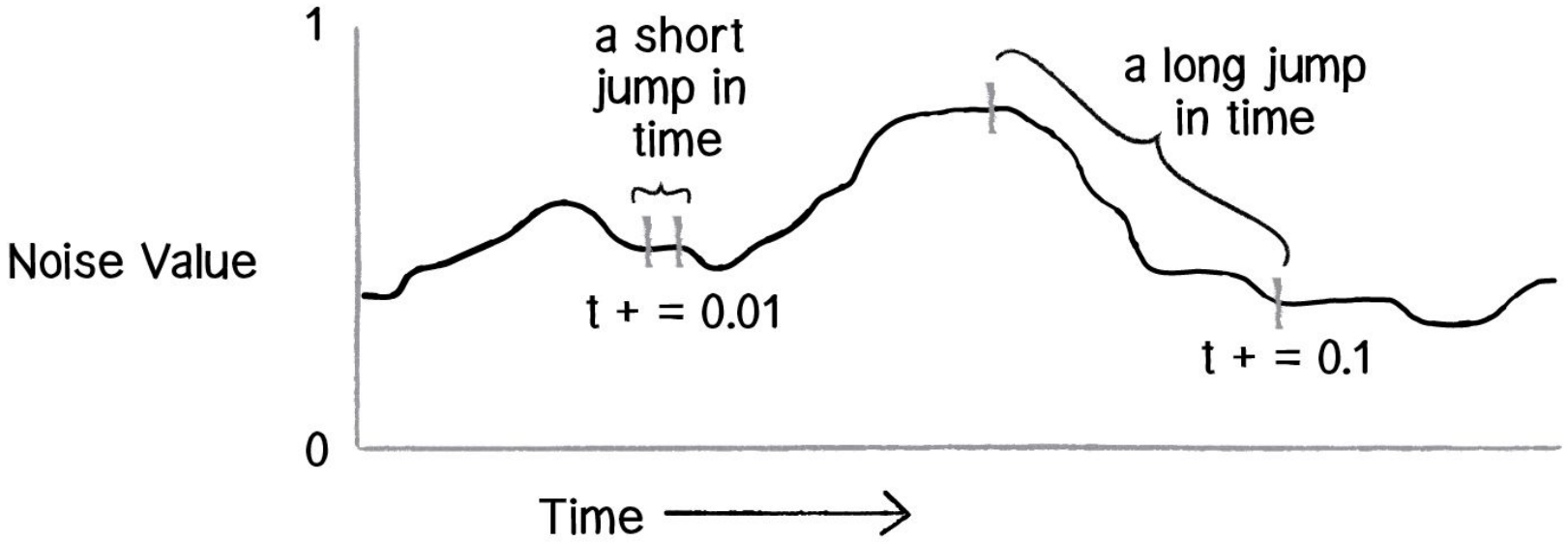
ofNoise()

Time	Noise Value
0	0.365
1	0.363
2	0.363
3	0.364
4	0.366

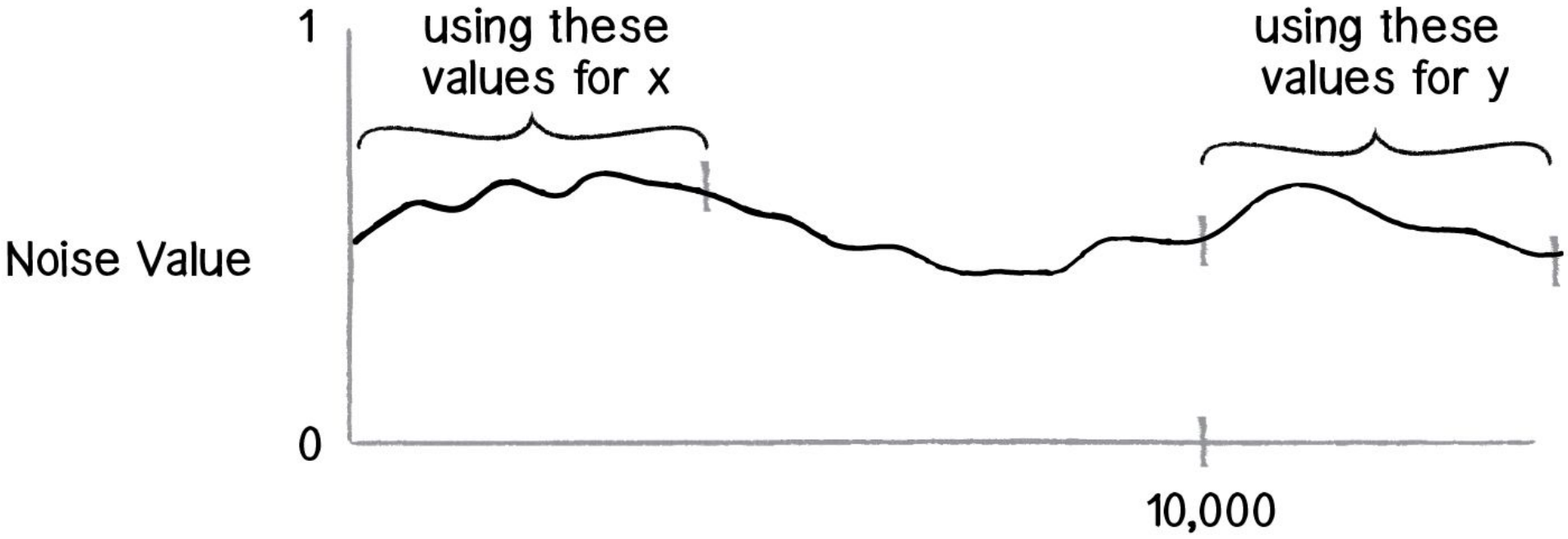
```
ofNoise(0)   = 0.365
ofNoise(1)   = 0.363
```

etc...

ofNoise()

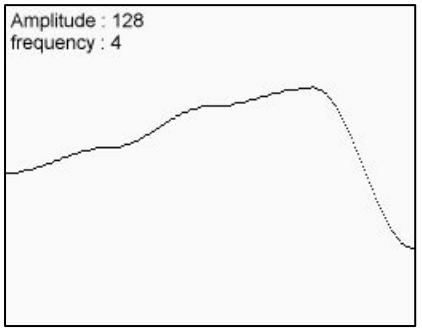

1
a short
jump in
time
a long jump
in time
Noise Value
t + = 0.01
t + = 0.1
0
Time

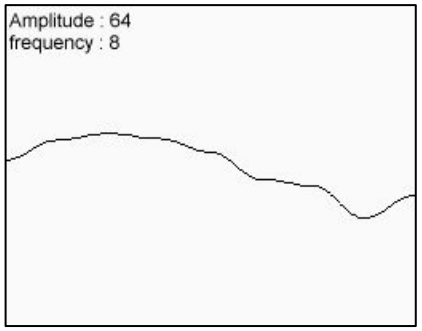
using different values on noise function

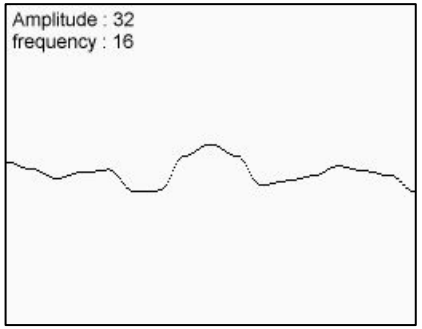


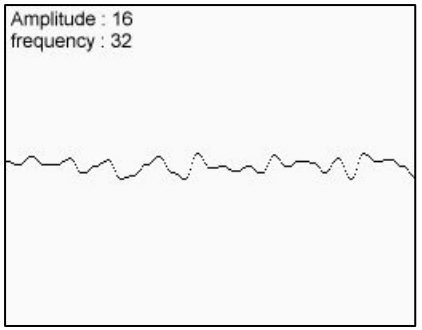
how is it made?

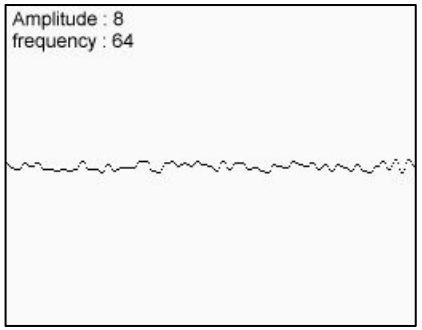
adding different noise waves together

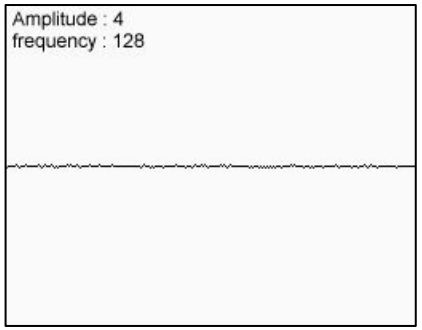




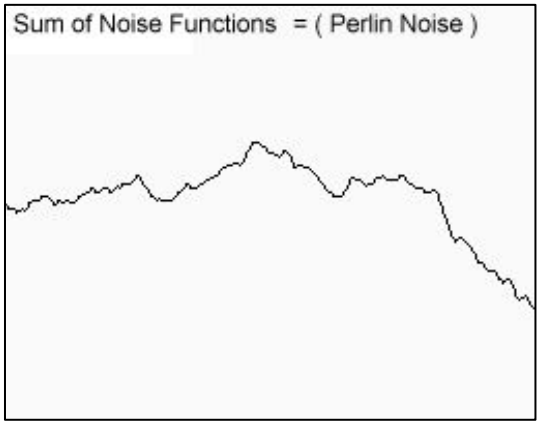




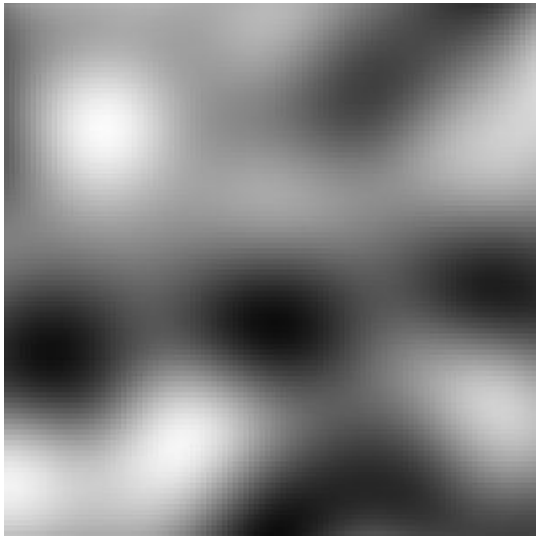




...you get Perlin noise



1D Noise



2D Noise

https://openframeworks.cc/documentation/math/ofMath/#show_ofNoise

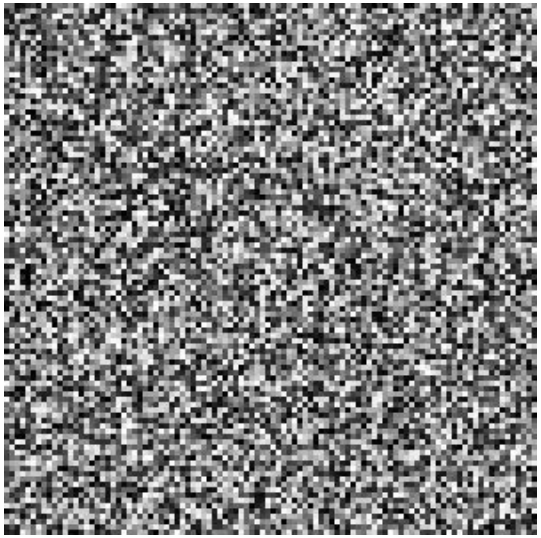
setting y coordinates in a line

Random

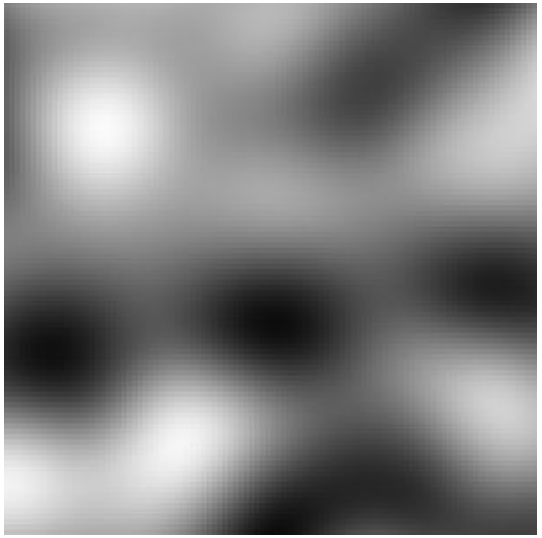
1D Noise

Sine

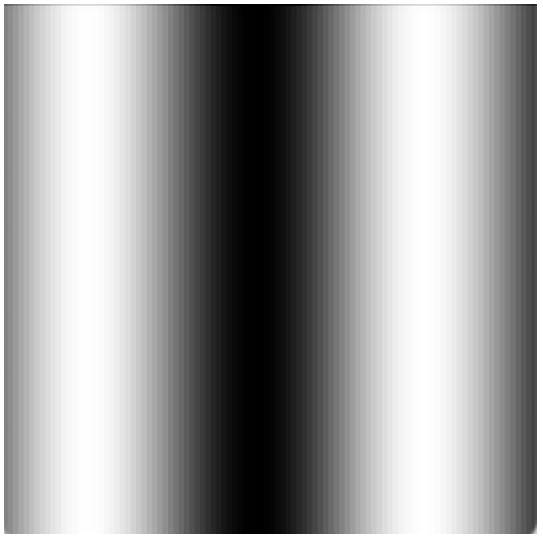
Review 2d noise



Random



2D Noise



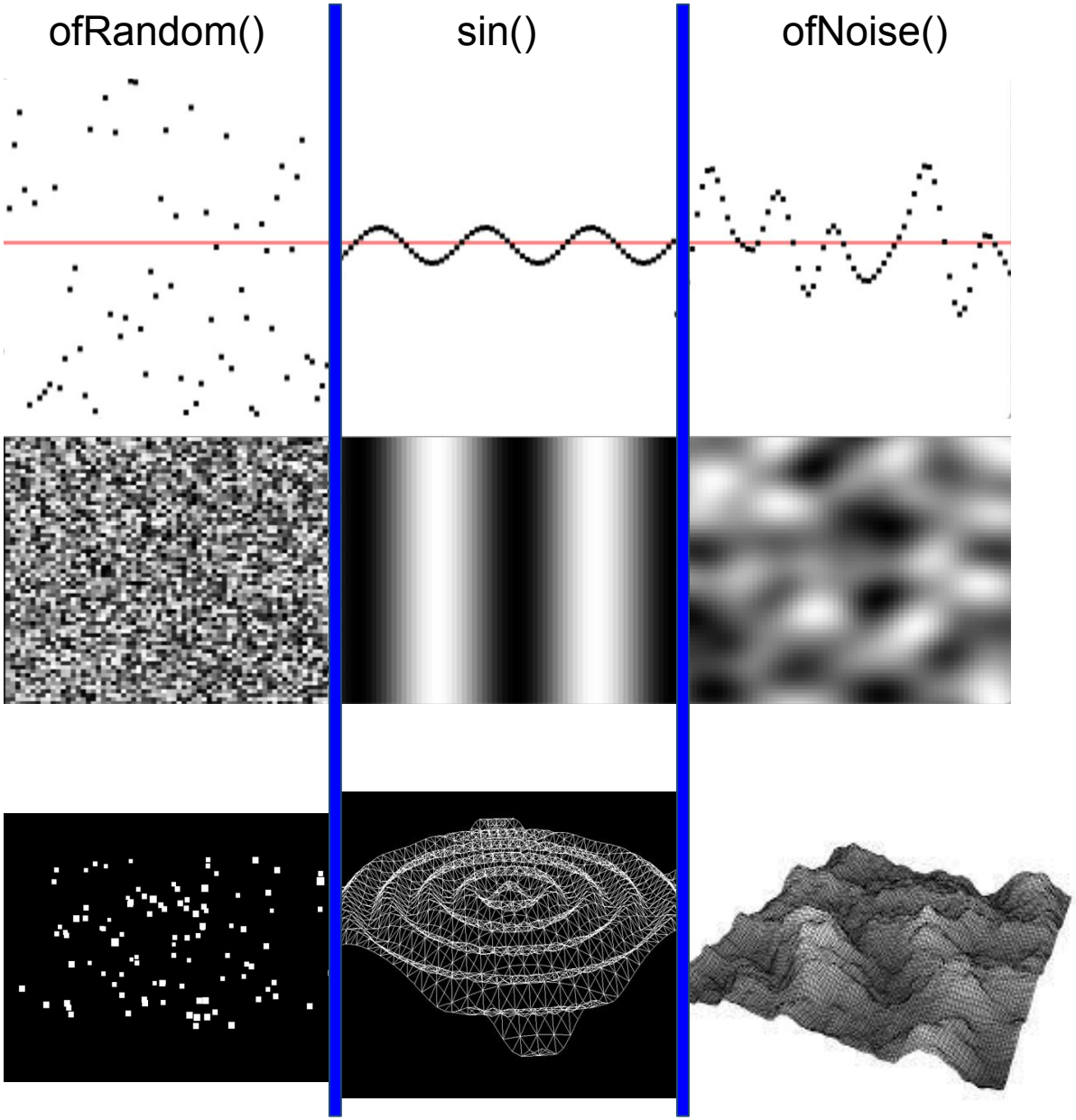
Sine

randomPixels
noisePixels
sinePixels

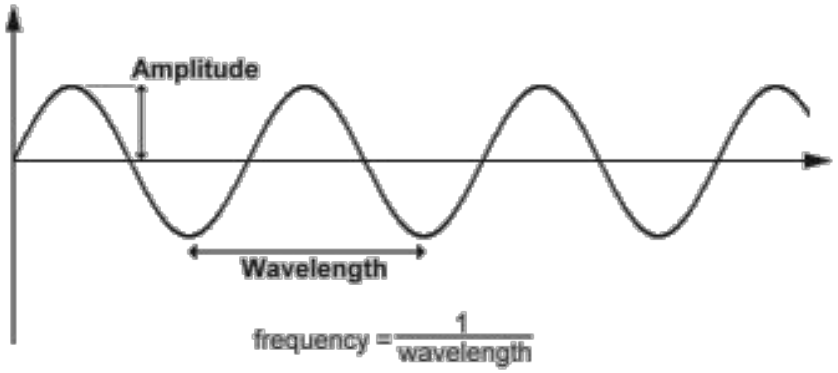
1 Dimensional Graph >

2 Dimensional: used for procedural textures >

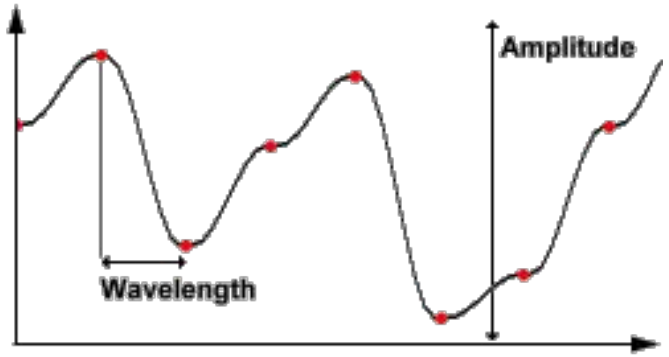
Many applications! >



terminology



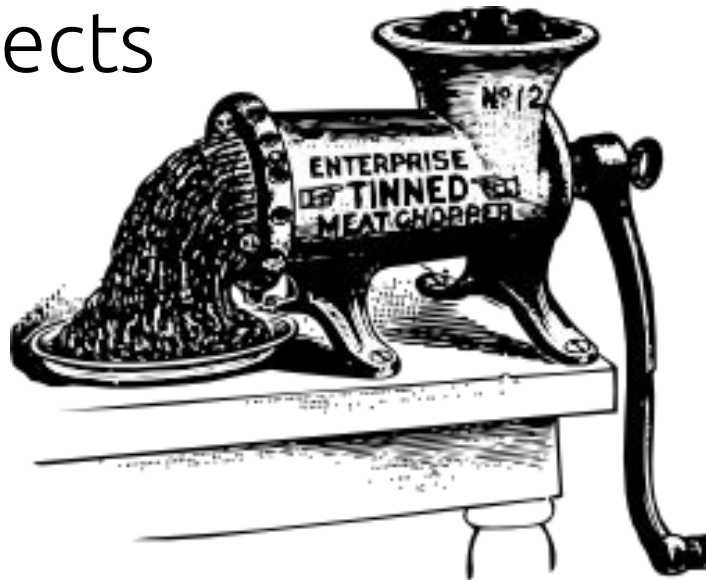
for a sine wave



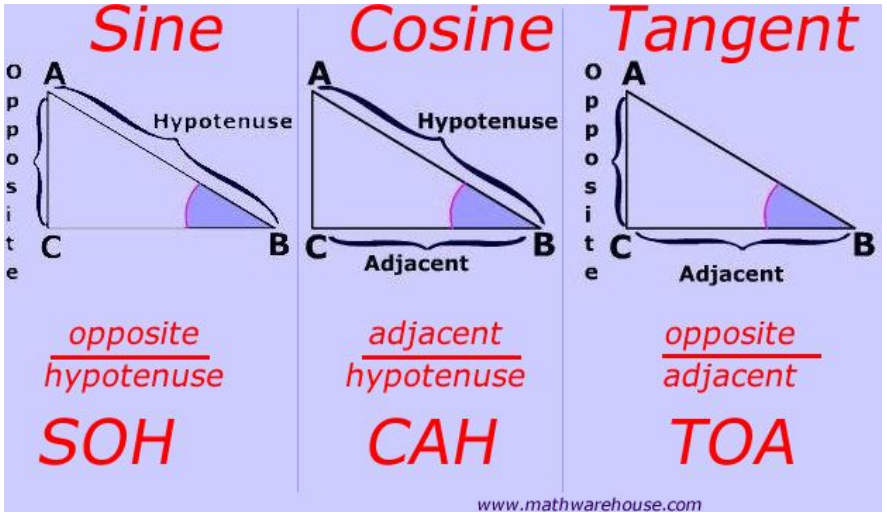
for a noise wave

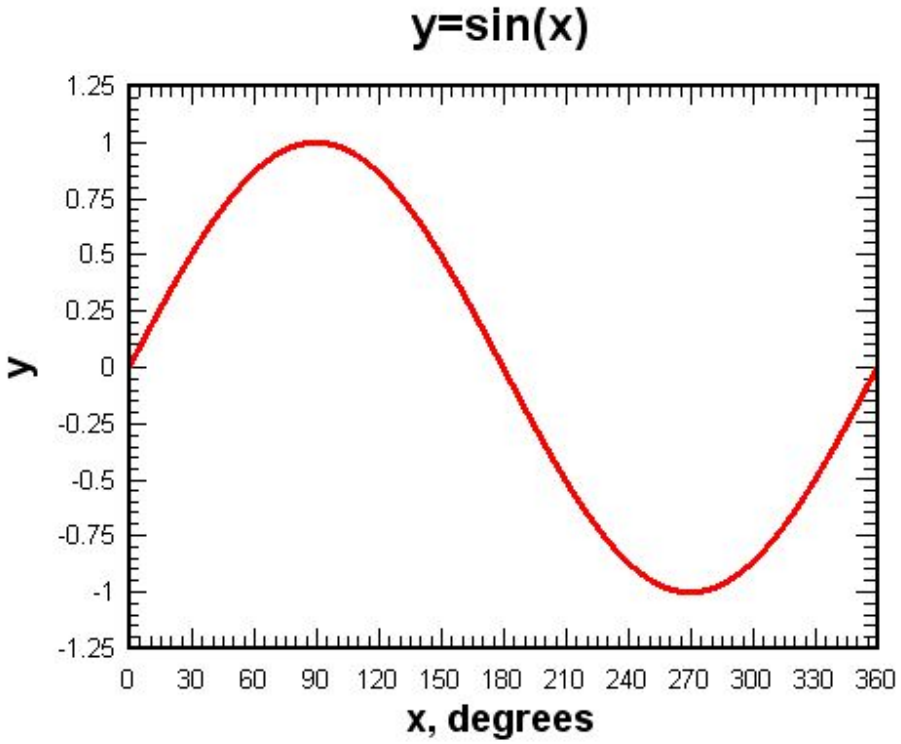
Sine as a number machine

- We feed in linear values, we can send in angles (degrees or radians) or hack it using a value that increases over time
- We get a smooth sequence of non-linear values
- They cycle between -1 and 1
- allows us to make oscillating objects



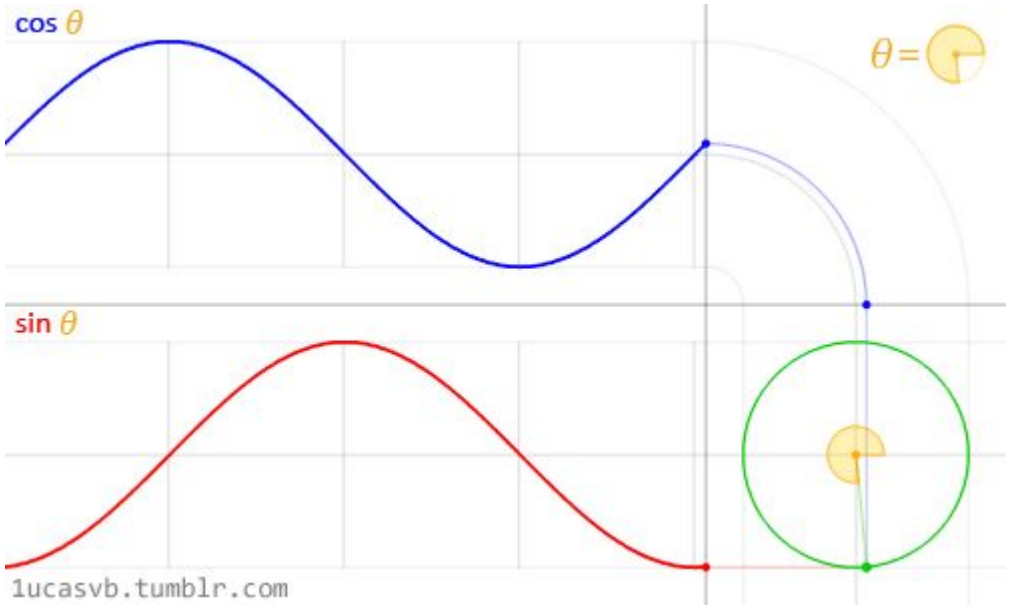
sine function

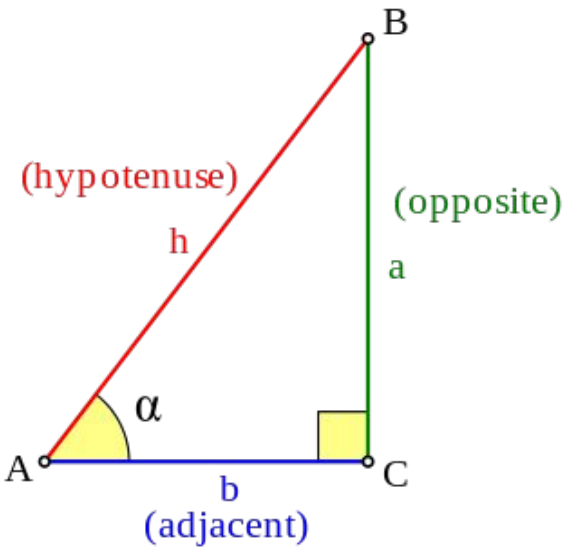


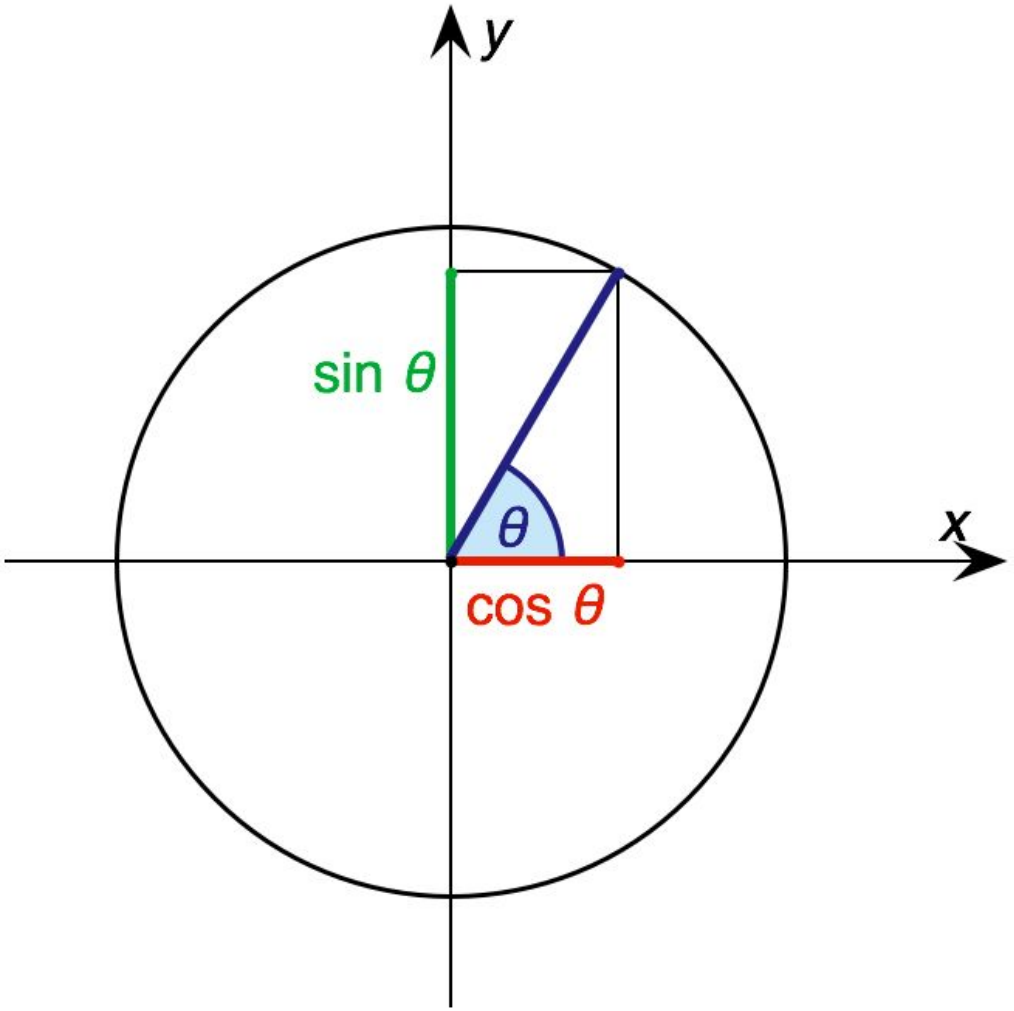


combine sin + cos to make circles

- `x = cos(ofDegToRad(angle)) * radius;`
- `y = sin(ofDegToRad(angle)) * radius;`




B
(hypotenuse)
h
(opposite)
a
α
A
C
b
(adjacent)


y
sin θ
θ
x
cos θ

Vector Math Review Slides

(if we need them, look at last week's slides for full detail)

We will be using & talking about vector math again after reading week, so if it feels like too much for now, feel free to ignore it for now.

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/

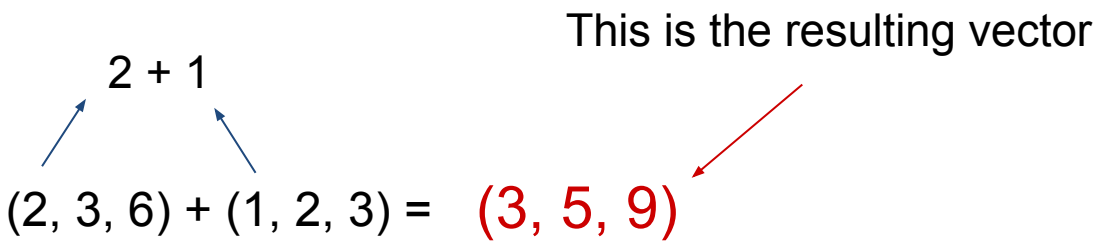
vec2 is for two values – xy

- 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

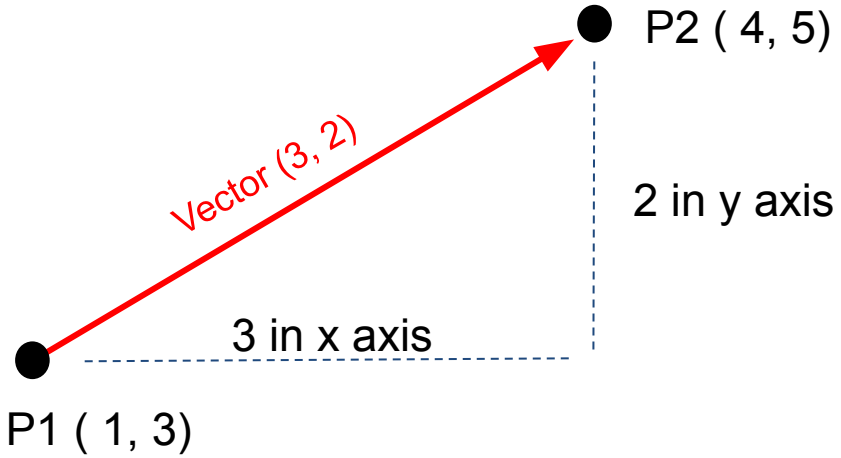
```
vec2 b = vec2(2.0, 3.0)*4;
```

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

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(redArrow.y, redArrow.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/