programming for artists and designers

Daniel Berio

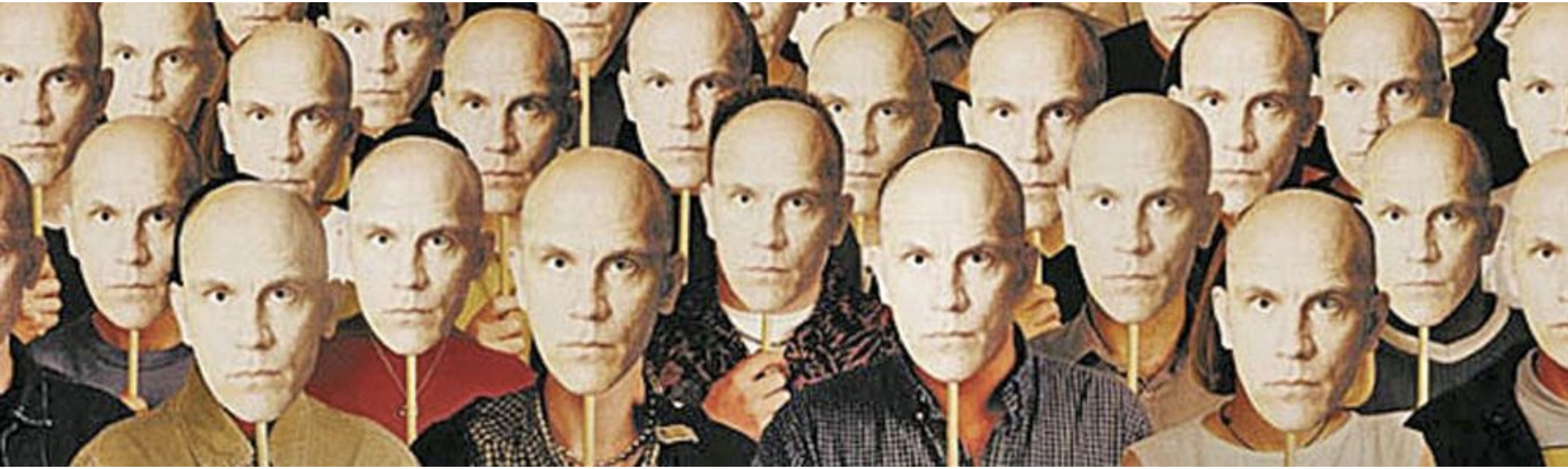
Mid-term Project

- **Week 4, Q&A next week:**
  - *Understanding the prompt:* look at some examples of work
  - *Understanding the design constraints:* rules and restrictions
  - Trying AxiDraw (hopefully)
- **Week 5, feedback on your idea from peers:**
  - Bring a code sketch
  - Written ideas / hand drawn sketches
- **Week 6, after reading week:**
  - In class presentation / hanging your work up on the wall in the Church
  - End of the week hand in code & short report

Session Summary

- Recap:
  - Conditionals
  - Transformations
- Loops
  - 1d loops
    - Generating curves
  - 2d loops
    - Generating “grids”
- Lab activities / at home:
  - Generative texture
  - Intro to generative art
  - Using randomness & probability

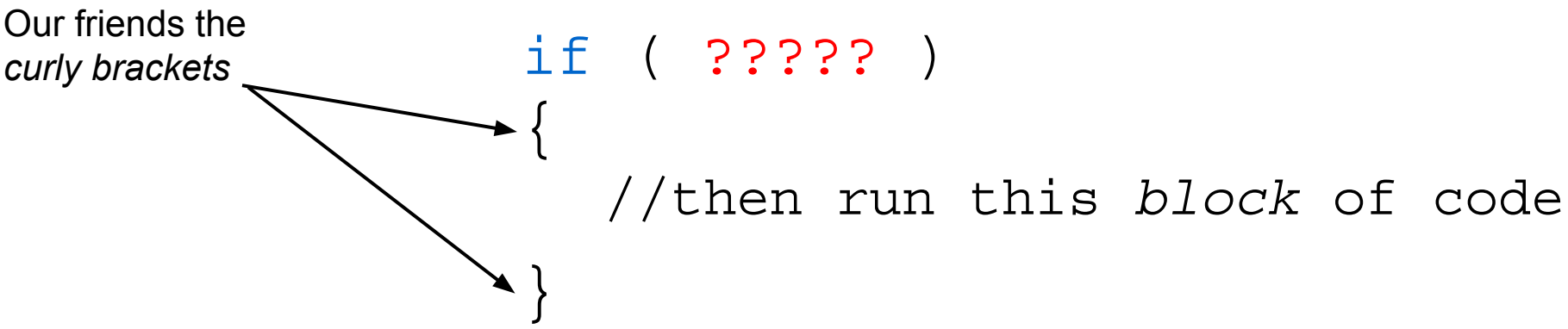
Repeat! Repeat! Repeat!



Recap: Control flow

CONTROL FLOW:
controlling *when* a
block of code is run

conditional statements



`if` within our ofApp.cpp

- we run some commands some of the time and not others
- different behaviours depending on conditions

```
void ofApp::draw()
{

    if(??????)
    {


    }

}
```

boolean statements

- Socrates is dead — TRUE
- donkeys fly — FALSE

- 15 greater than 20 — FALSE
- 5 equals 5 — TRUE

relational operators

Operator	Relation	Notes
`<`	Less than	E.g. `5 < 10` is `true`
`>`	Greater than	E.g. `5 > 10` ia `false`
`<=`	Less or equal to	E.g. `10 <= 10` is `true`
`>=`	Greater or equal to	E.g. `12 >= -10` is `true`
`==`	Equal	E.g. `a == b` is `true` if `a` and `b` have the same value
`!=`	Not equal	E.g. `a != b` is `false` if the above is `true`

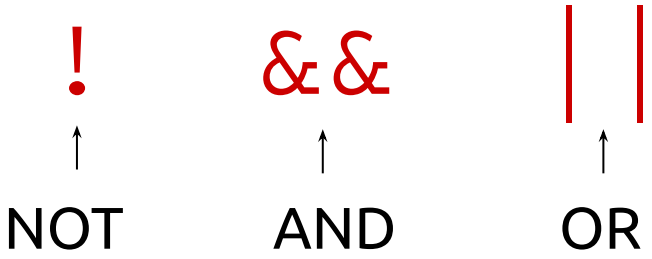
Conditional statements - `if / else if`

```
if ( raining )
{
   - take car     // executes this if raining
}
else if ( sunny )
{
   - take bicycle       // this if sunny
}
```

Conditional statements - `if / else`

```
if ( raining )
{
   - take car     // executes this if raining
}
else if ( sunny )
{
   - take bicycle       // this if sunny
}
else
{
   - stay home        // this for every other case
}
```

Logical operators:




```
if ( !raining )  // IF it's NOT raining
{
     - take bicycle
}
```

```
if ( raining && you have license ) // IF raining AND you drive
{
    - take car
}
```

```
if (raining || snowing) // IF it's raining OR snowing
{
    - wear jacket
}
```

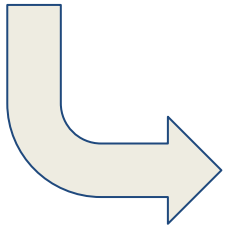
One use of `if` today: Save output to SVG/PDF

- Scalable Vector Graphics (SVG) and Portable Document Format (PDF) are **vector graphics file** formats.
  - Can open files with software such as:
    - Adobe Illustrator
    - Affinity Designer
    - Inkscape (open source)
- They will enable you to output graphics that can be printed at any size, and in certain cases drawn with a machine
- Download template code

Save output to SVG/PDF

This gets called when user presses a key

```
void ofApp::keyPressed(int key){
  if (key == ' ') {
    saving = true;
  }
}
```



Next in `ofApp::draw....`

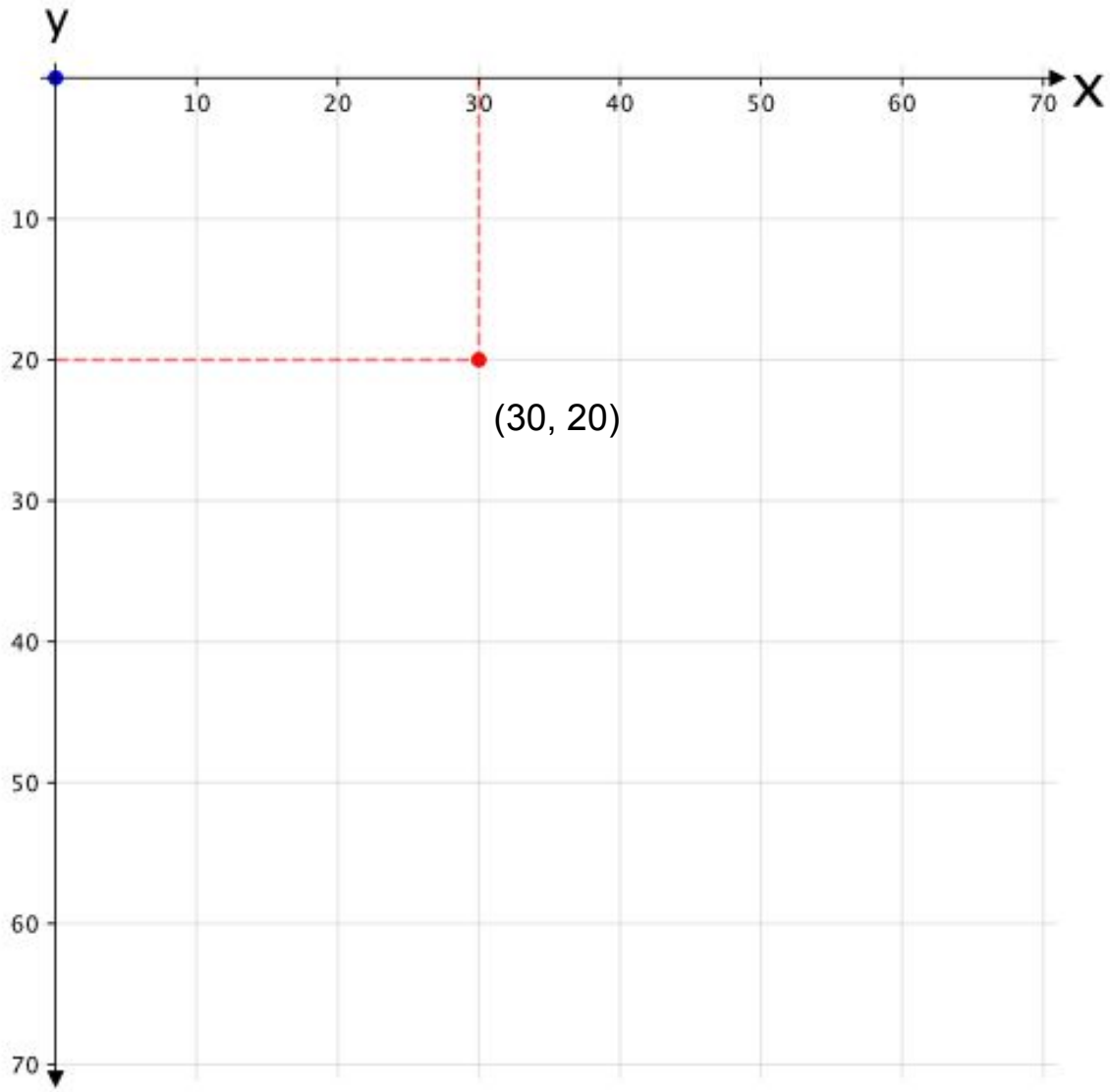
Before any drawing happens:

```
if (saving) {
    // the file will end up in
     // the ./bin/data directory
    ofBeginSaveScreenAsSVG("mySvg.svg");
}
```

After drawing happened

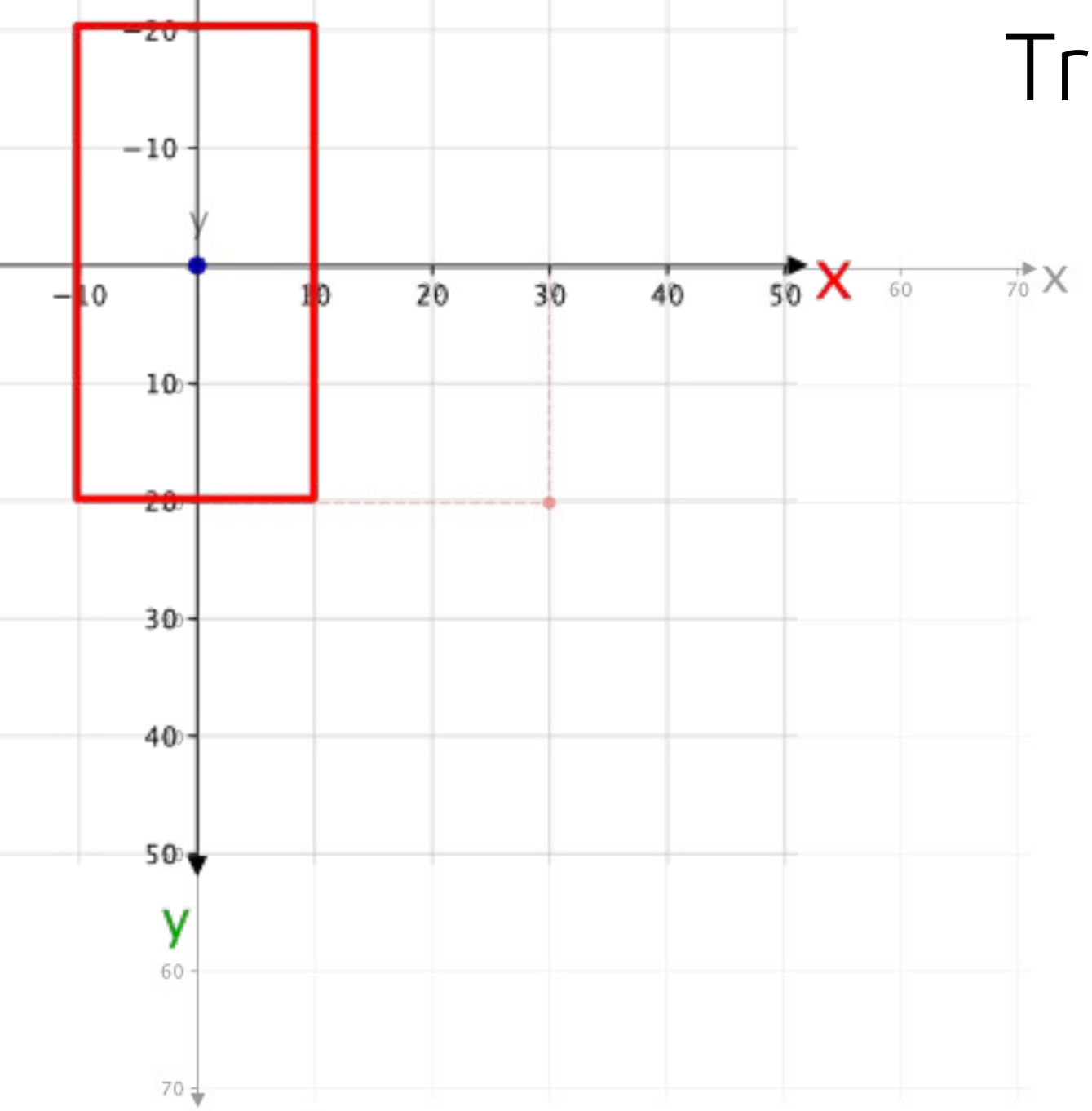
```
if (saving) {
    ofEndSaveScreenAsSVG();
    cout << "Saved svg" << endl;
    saving = false; // reset the saving variable
 }
```

More recap: window coordinate system



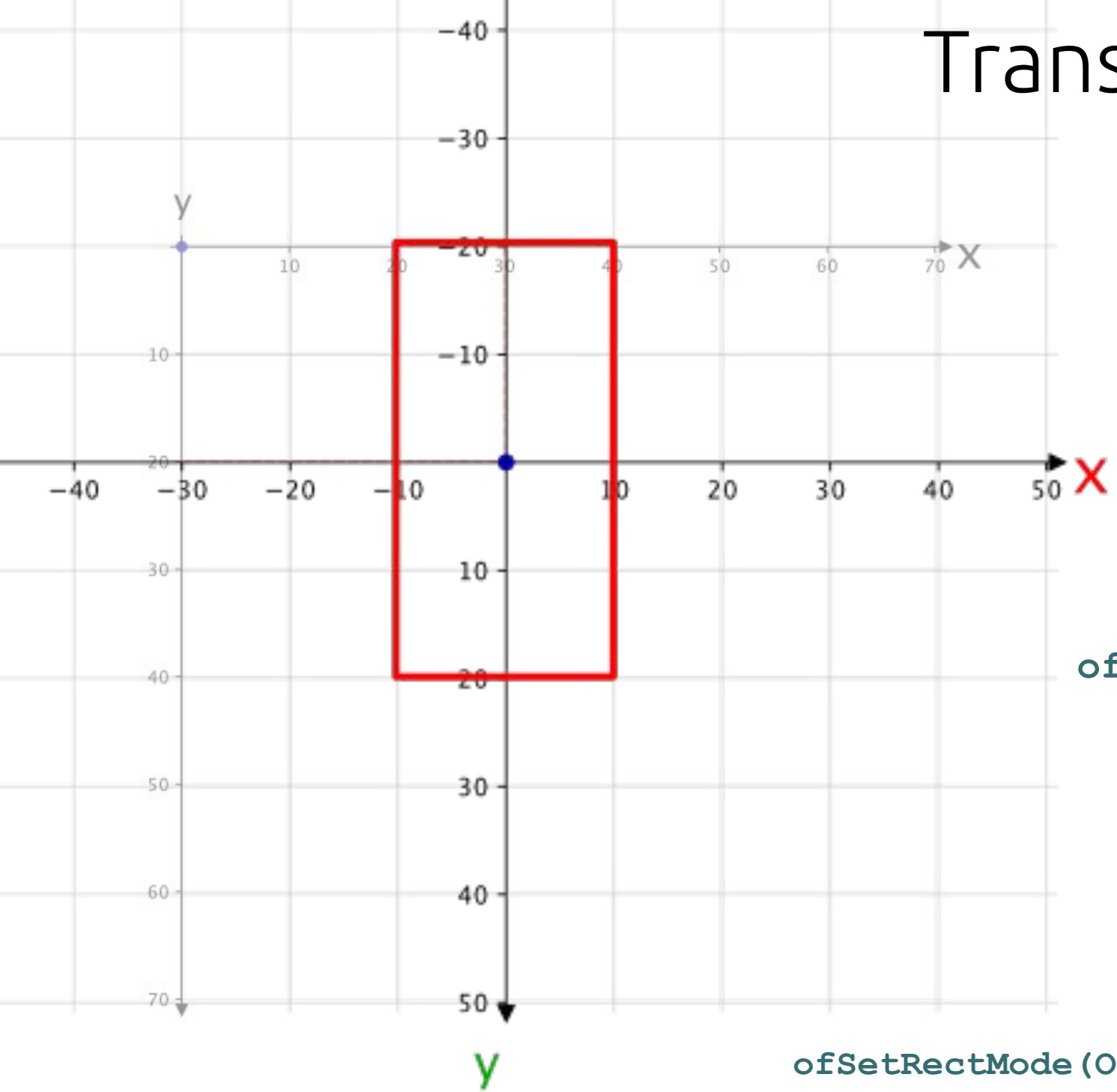
Every time `ofApp::draw` is called

Transformations




```
ofSetRectMode(OF_RECTMODE_CENTER);
ofDrawRectangle(0, 0, 20, 40);
```

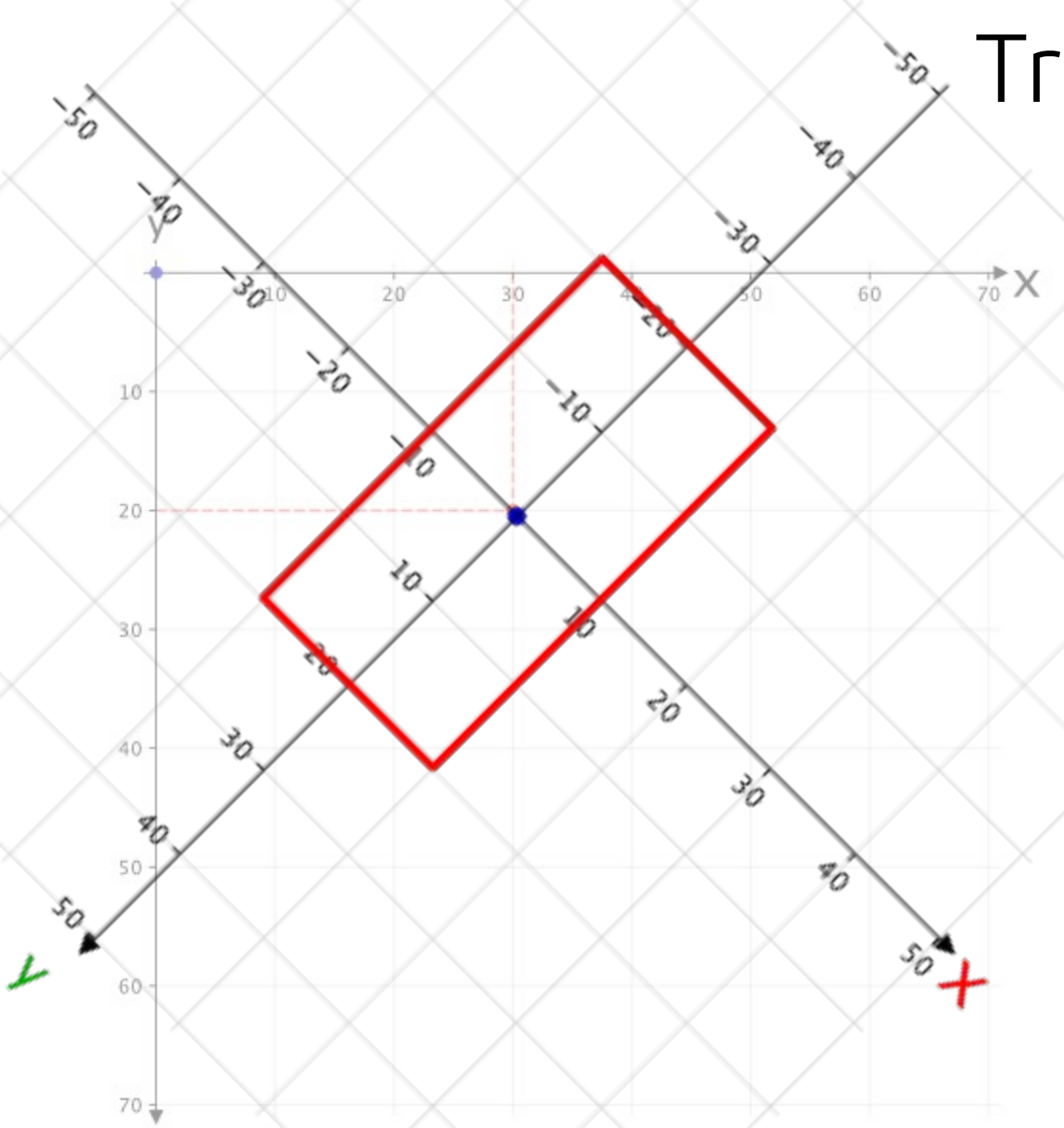
Transformations




```
ofTranslate(30, 20);
```

```
ofSetRectMode(OF_RECTMODE_CENTER);
ofDrawRectangle(0, 0, 20, 40);
```

Transformations

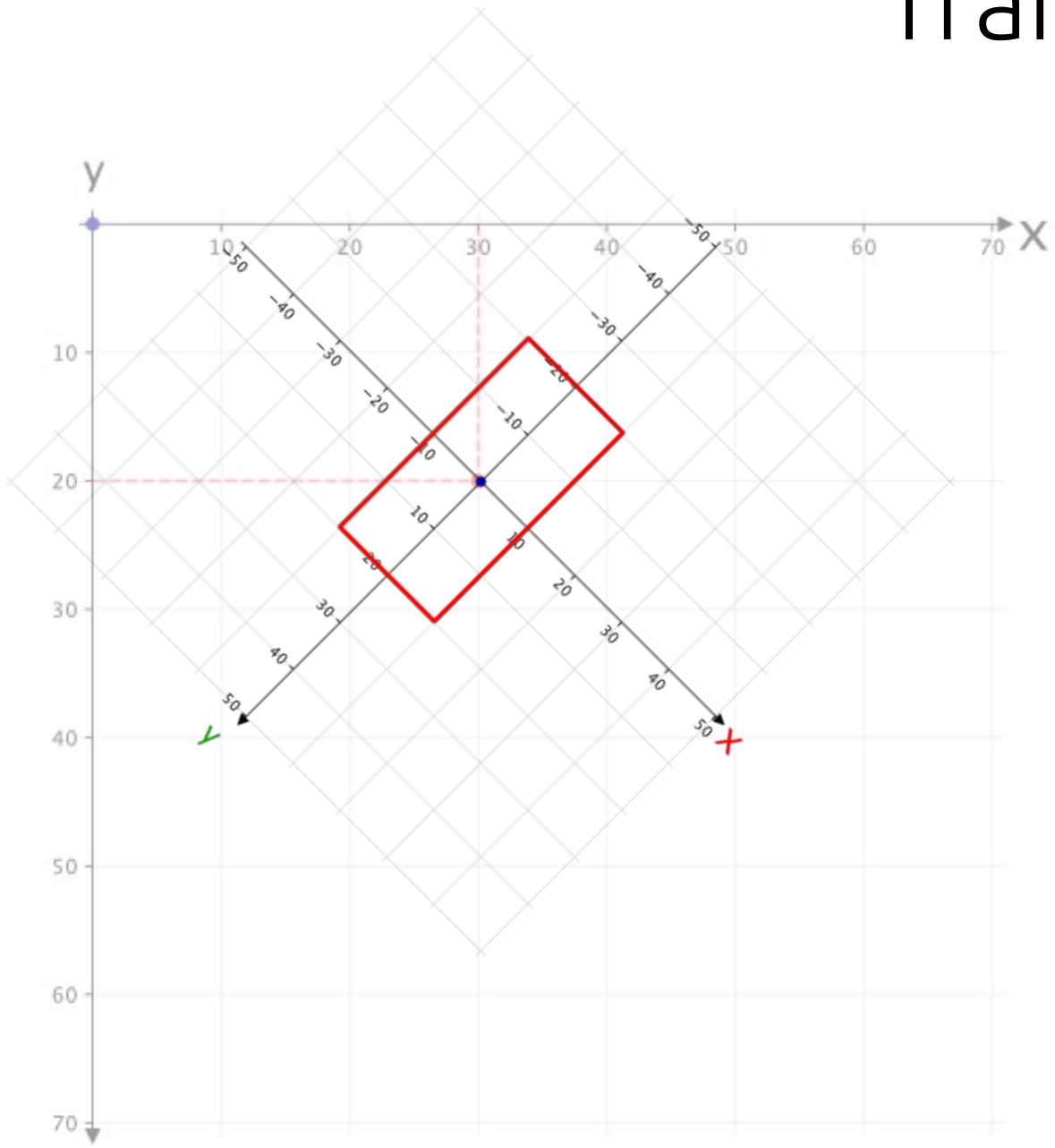


Think of them as changing the transforming the current coordinate system

```
ofTranslate(30, 20);
ofRotateDeg(45);
```

```
ofSetRectMode(OF_RECTMODE_CENTER);
ofDrawRectangle(0, 0, 20, 40);
```

Transformations

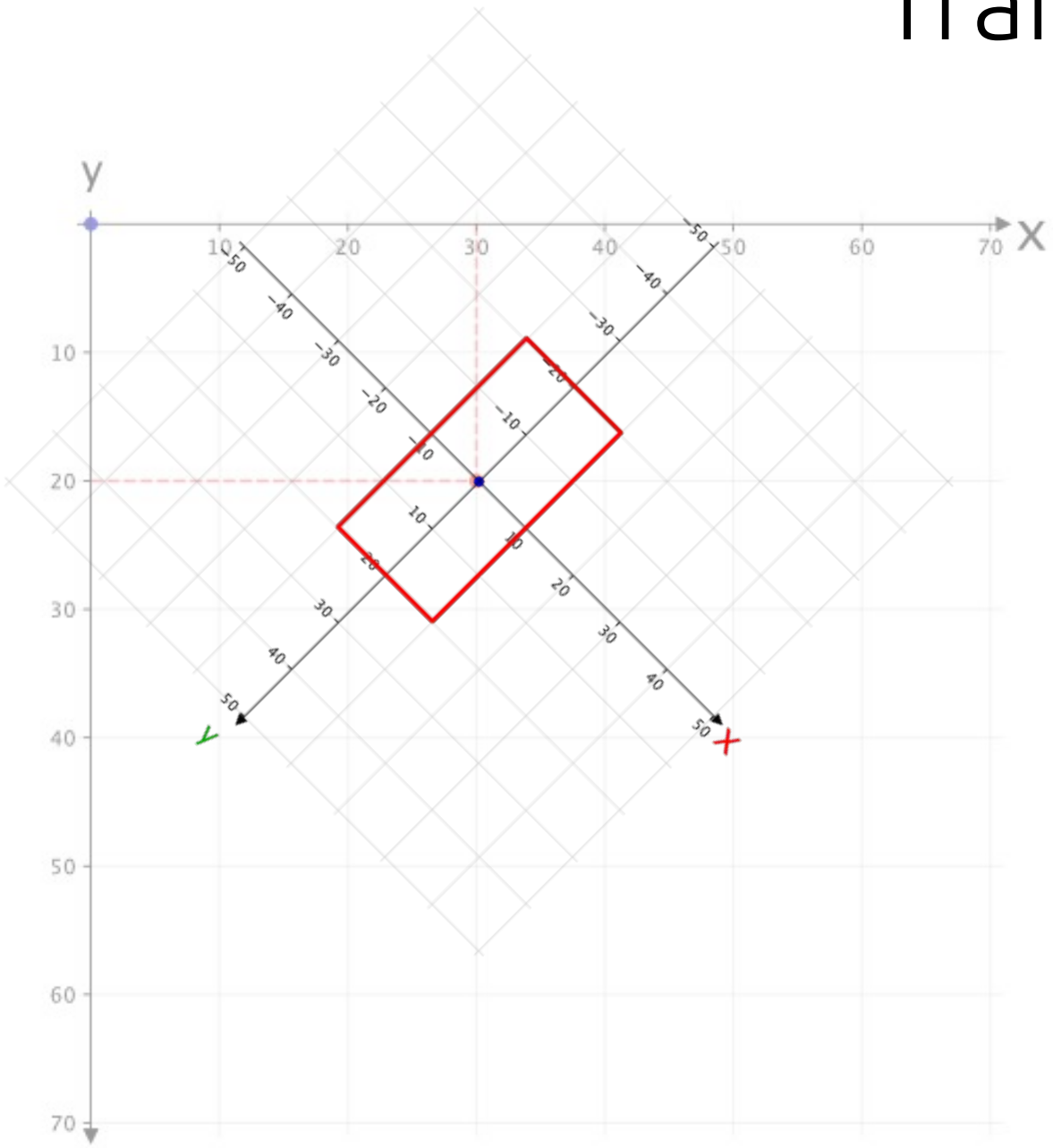


Think of them as changing the transforming the current coordinate system

```
ofTranslate(30, 20);
ofRotateDeg(45);
ofScale(0.5);
```

```
ofSetRectMode(OF_RECTMODE_CENTER);
ofDrawRectangle(0, 0, 20, 40);
```

Transformations

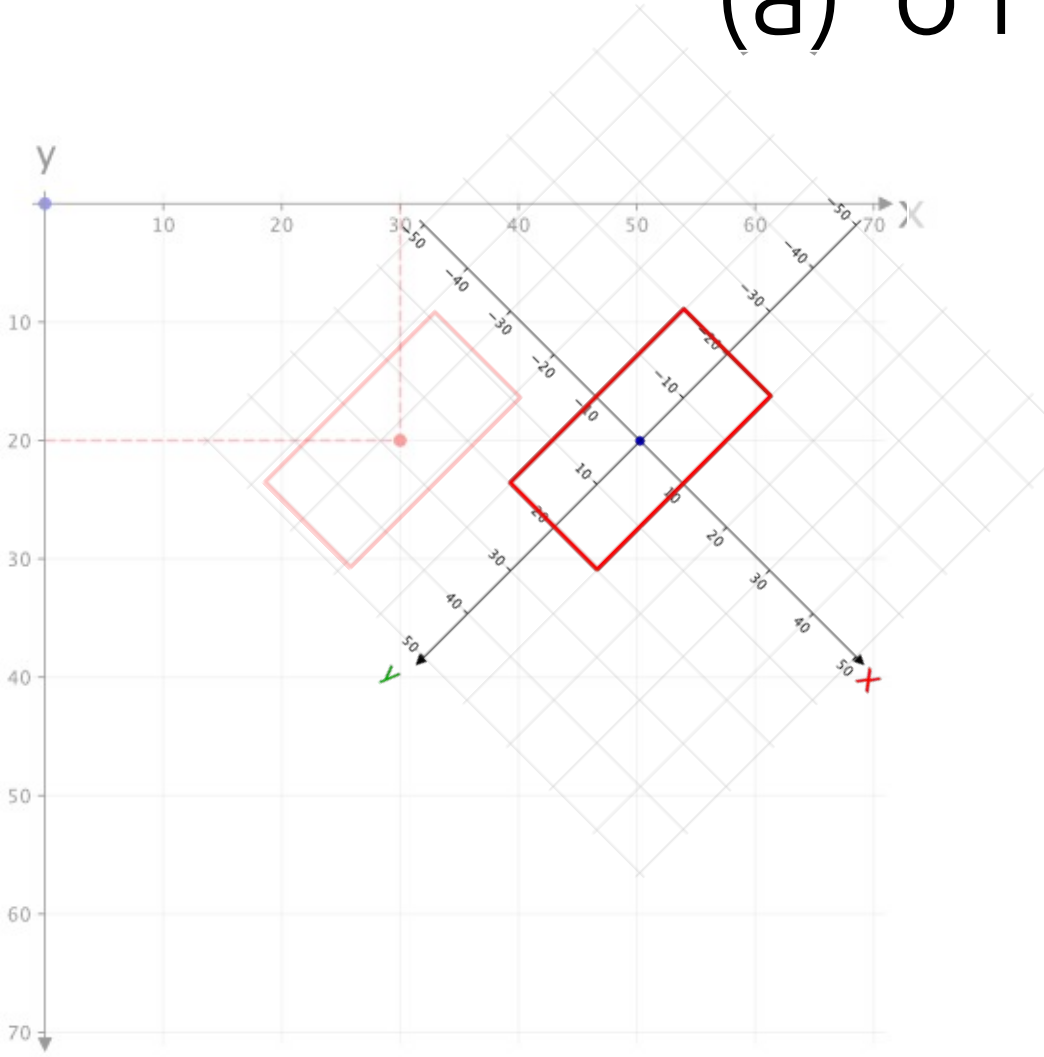



```
ofTranslate(30, 20);
ofRotateDeg(45);
ofScale(0.5);
ofTranslate(20, 0);
```

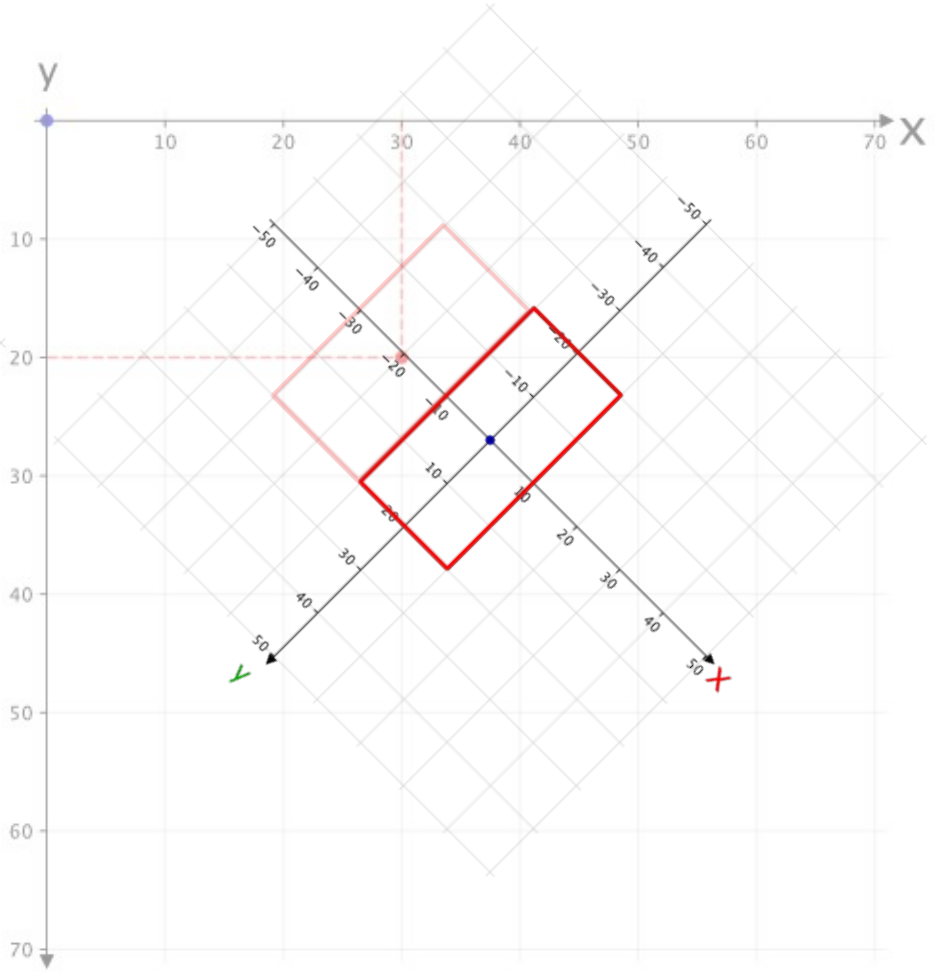
?

```
ofSetRectMode(OF_RECTMODE_CENTER);
ofDrawRectangle(0, 0, 20, 40);
```

(a) or (b) ?

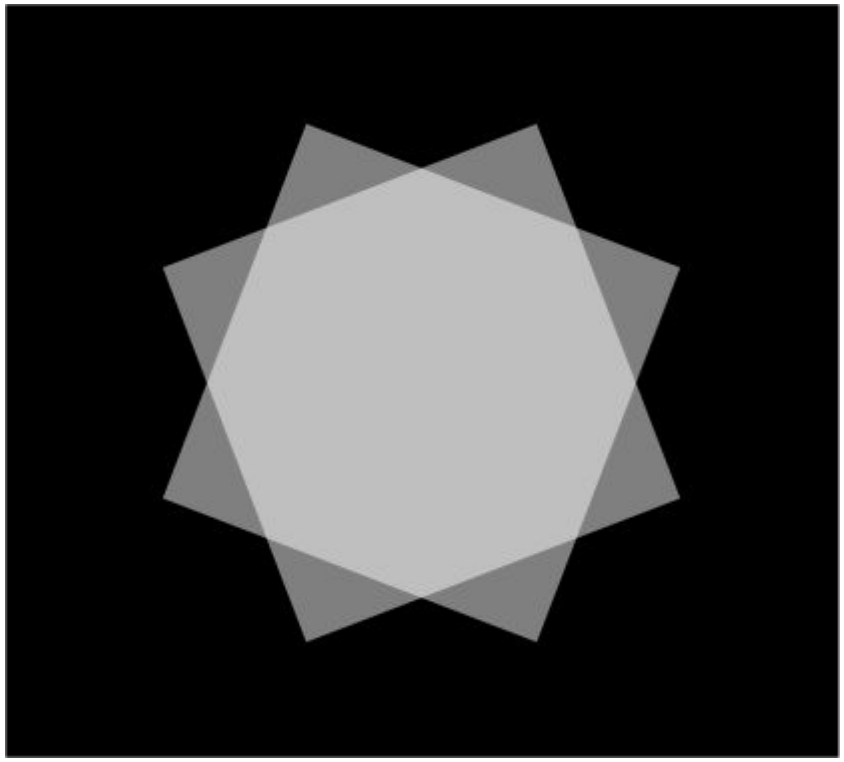


(a)



(b)

```
ofTranslate(30, 20);
ofRotateDeg(45);
ofScale(0.5);
ofTranslate(20, 0);
```



New: push/pop matrix

- Transformations are reset at each call to draw
  - (origin at top left)
- Say we want to have 2 (or more) rectangles rotating in different directions, how do we do it?
  - `ofPushMatrix()` and `ofPopMatrix()`
- Like “scope” for transformations
  - `ofPushMatrix()` saves the current transformation
  - `ofPopMatrix()` resets the last saved one ?

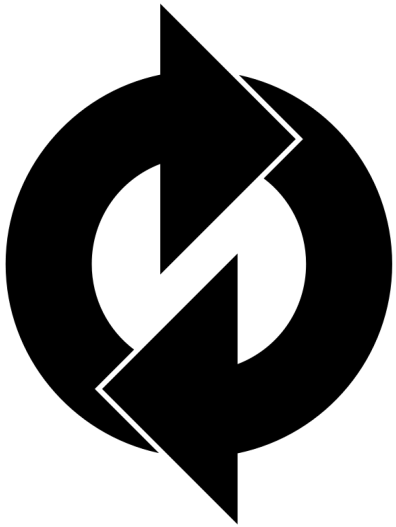
```
ofPushMatrix();
ofRotateDeg(rot1);
// draw stuff ...
ofPopMatrix();

ofPushMatrix();
ofRotateDeg(rot2);
// draw stuff
ofPopMatrix();
```

pushMatrixExample


introduction
to loops

Loops

- we use loops since we learned about animation
- `draw()` is a loop, but:
  - it never stops
  - there is only one `draw()` in each sketch
  - it changes elements on screen
  - it's slow (as fast as our framerate, e.g. 60 fps)



while

"While it's raining I am going to stay home"

```
while ( raining )   // evaluates statement
{
   - stay home
}
```

whileLoopExample

`while` (behind the OF hood)

"While user does not press ESC draw

```
while ( !escPressed )     // evaluates statement
{
   draw()
}
```

whileLoopExample

Loops - while()

```
void ofApp::draw(){
  int circleX = 0;

  while (circleX < ofGetWidth()){

    ofDrawCircle(circleX, ofGetHeight()/2, 10);
    circleX = circleX + 50;

  }
}
```

initialization

evaluating the conditional

updating the variable

whileLoopExample

Loops - while()

```
void ofApp::draw(){
  int circleX = 0;

  while (circleX < ofGetWidth()){

    ofDrawCircle(circleX, ofGetHeight()/2, 10);
    circleX += 50;

  }
}
```

initialization

evaluating the conditional

updating the variable
(another syntax)

whileLoopExample

Loops - while()

```
void ofApp::draw(){
  int circleX = 0;

  while (circleX < ofGetWidth()){

    ofDrawCircle(circleX, ofGetHeight()/2, 10);
    circleX++;
  }
}
```

initialization

evaluating the conditional

updating the variable
(yet another syntax,
increment by 1)

whileLoopExample

Loops – `while`

if the conditional is always true, the program is going to run forever

Loops - for()

initialization

evaluating the conditional

updating the variable

```
void ofApp::draw(){

  for (int circleX=0; circleX < ofGetWidth(); circleX += 50){
    ofDrawCircle(circleX, circleY, 10);
  }

}
```

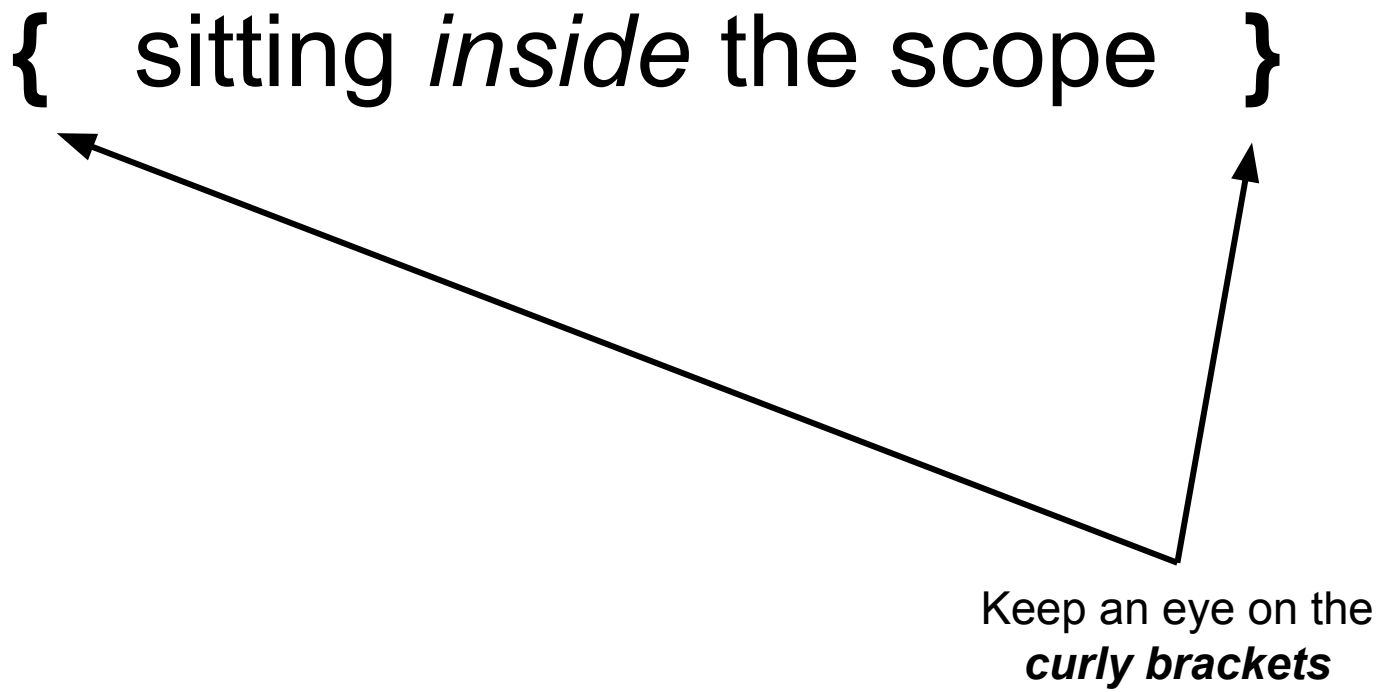
forLoopExample

Loops - for()

```
void ofApp::draw(){

  for (int circleX=0; circleX < ofGetWidth; circleX += 30){
    ofDrawCircle(circleX, circleY, 20);
  }

}
```

Can we make an observation about the new *syntax* we see?

Syntax is what is used to refer to the way things are written, e.g. using commas between our parameters.


{ sitting *inside* the scope }
Keep an eye on the
curly brackets

Loops - for

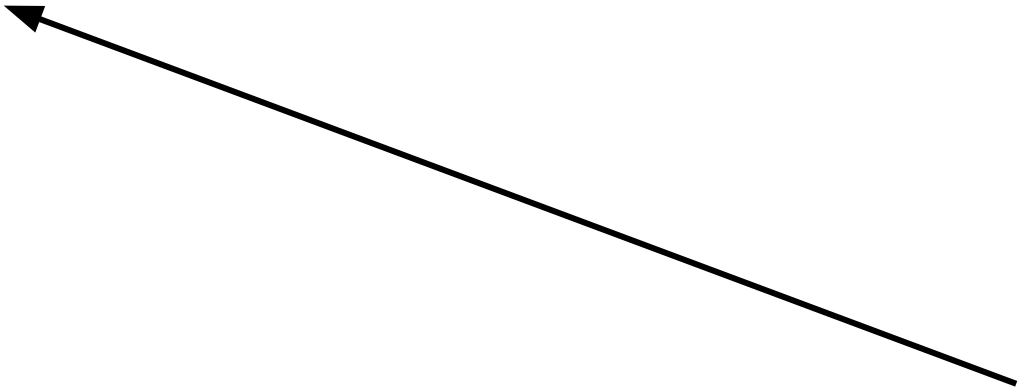
- updating the variable happens at the end of each cycle
- the variable that is initialized is local (it's known only within the loop, we can't use it outside)
- we can build a loop within loop if we want to add extra dimensions
- It is a convention to use variables named **i, j, k** as indices in for loops (in order from outer to inner)

```
for (int i = 0; i < 10; i++)
{
    for (int j = 0; j < 10; j++)
    {
        // code to execute
    }
}
```

nestedForLoop

```
if(...) {
    while(...) {
        ofDrawCircle(...)
    }
} // GOOD :)
```

```
if(...) {
while(...) {
ofDrawCircle(...)
    }
} // BAD
```



Keep an eye on the
Indentation:

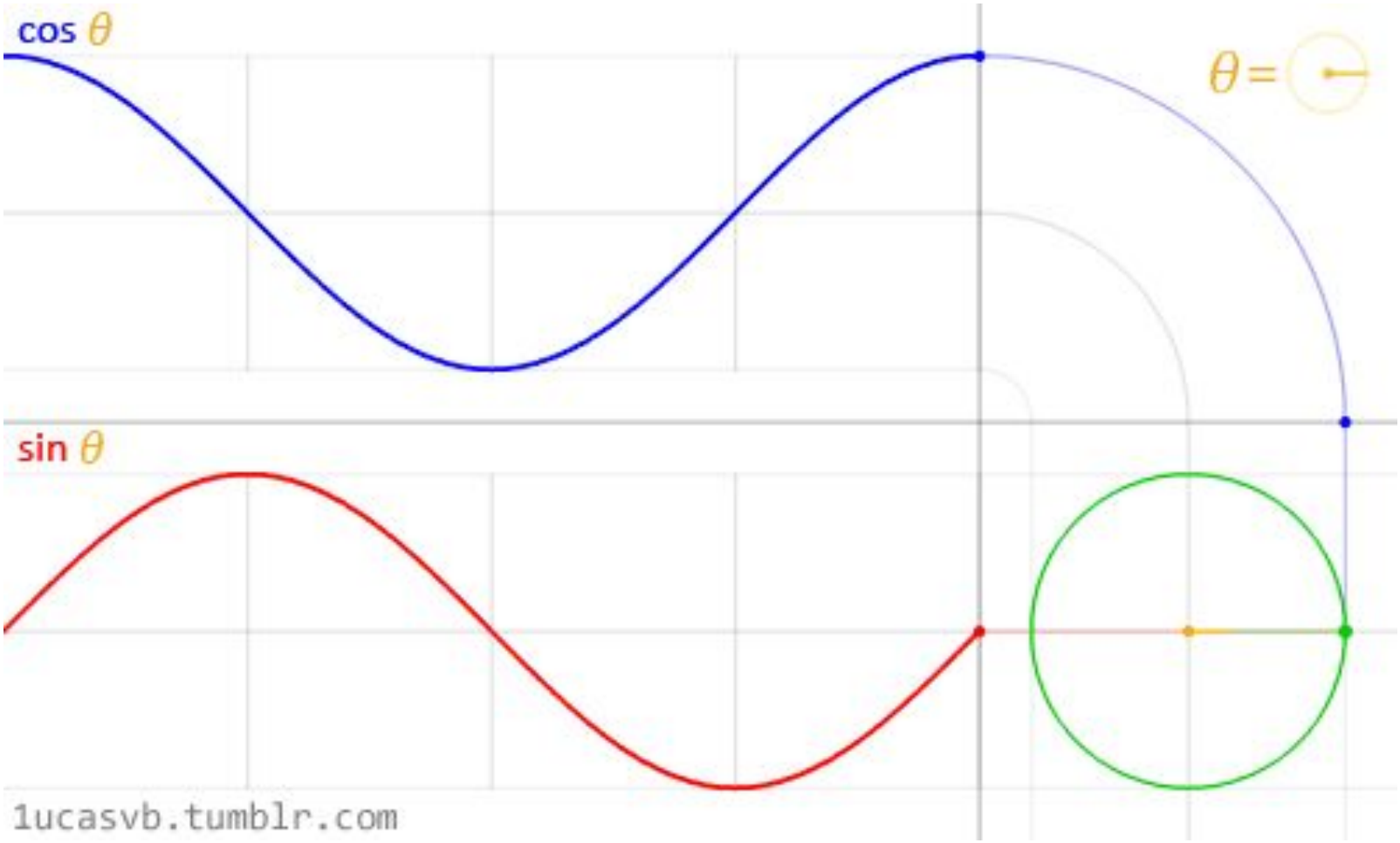
- ***Indent after {***
- ***Unindent after }***

scope

```
int x = 2; //in the header file

void ofApp::setup()
{
    int x = 3;
    cout << x << endl;              // ???
}

void ofApp::draw()
{
     cout << x << endl;             // ???
}
```

scope (cont.)

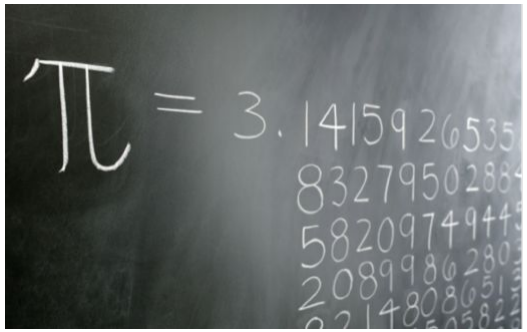
```
int x = 2;

void ofApp::setup()
{
    int x = 3;
    cout << x << endl;                    // ???
}

void ofApp::draw()
{
    for (int x = 5; x < 10; x++)
    {
        cout << x << endl;                // ???
    }
}
```




```
float x = cos(theta)*radius;
float y = cos(theta)*radius;
```

$$x = r\cos(t)$$

$$y = r\sin(t)$$



$$\tau = 2\pi = 6.28318530717958 6 \ldots$$

Constant provided in OF for π

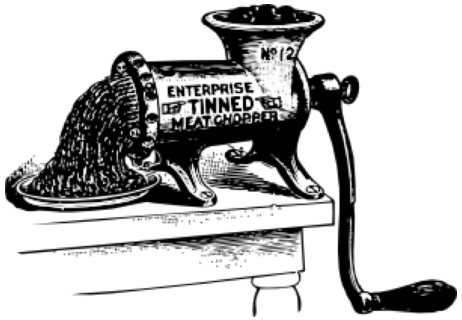
```
float tau = PI*2; // or

float tau = TWO_PI;
```

Constant provided in OF for 2π

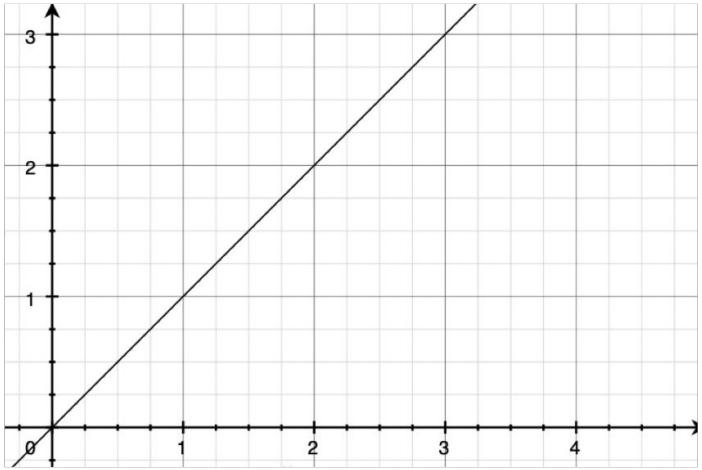
An argument for “TAU” instead of “PI”

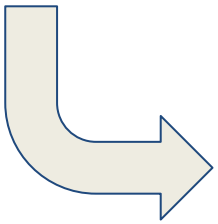
drawCircle

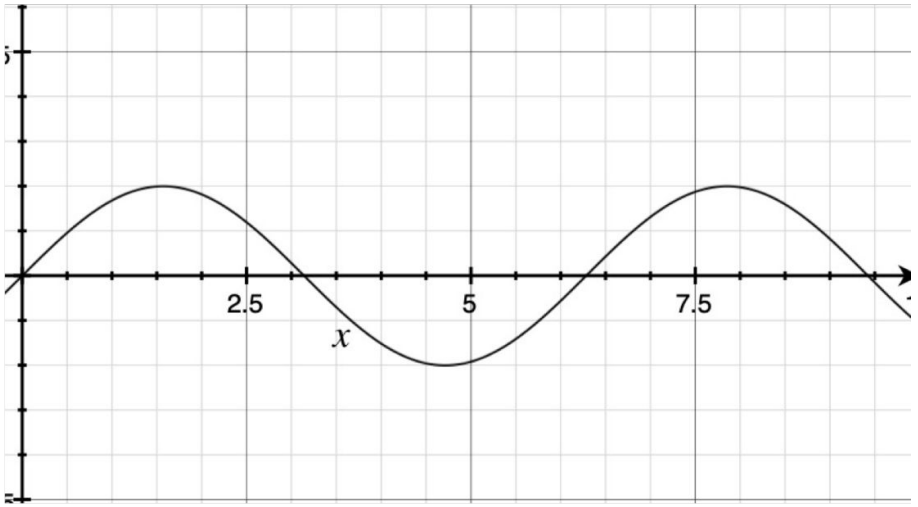
The sine function



input Linear number sequence

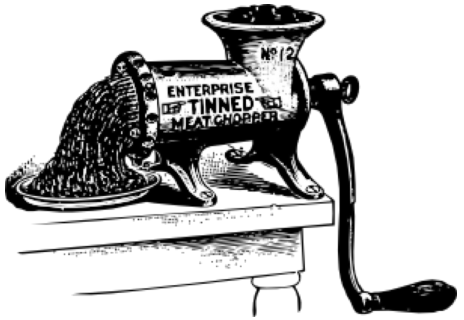


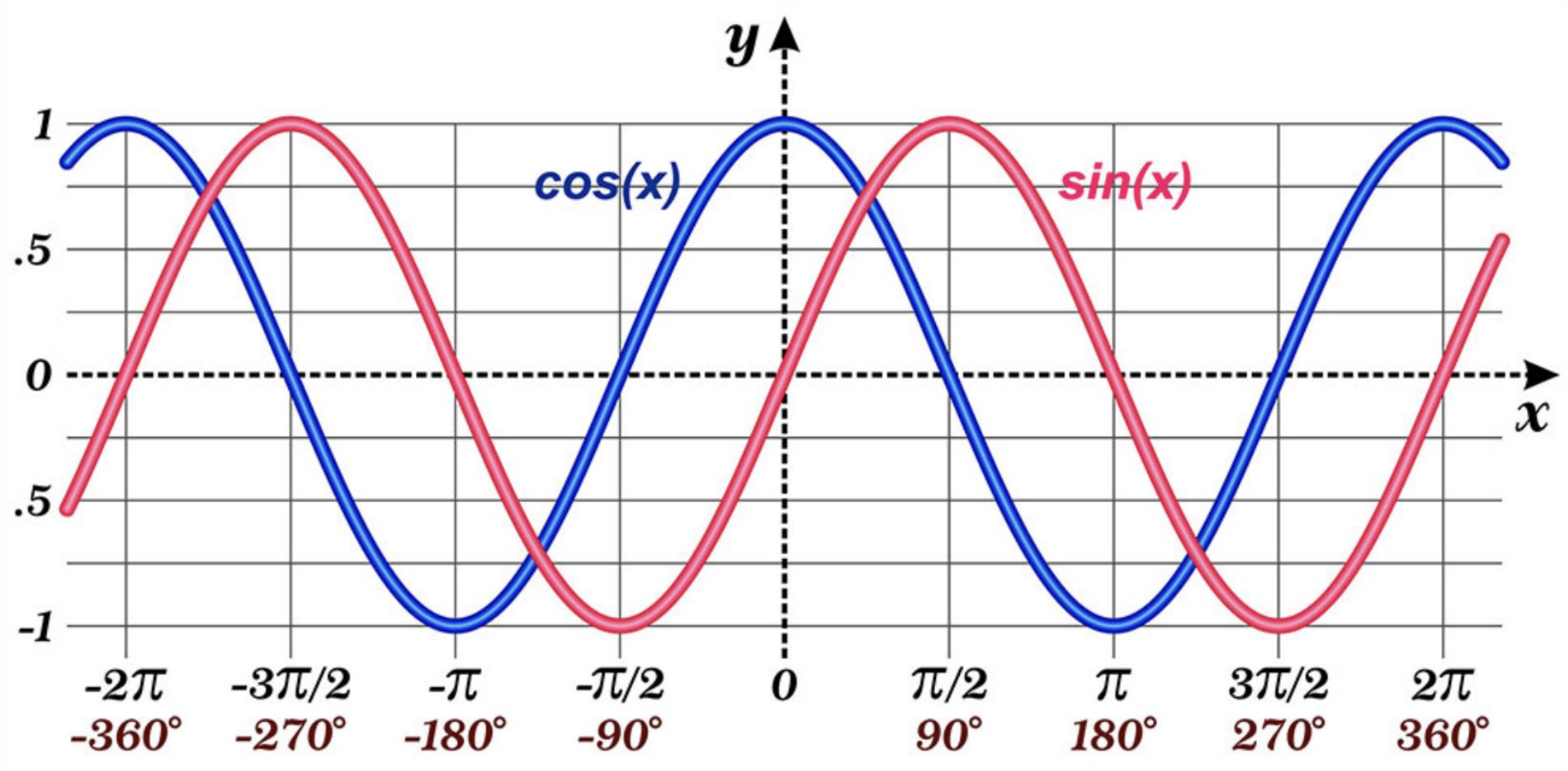




output Oscillating sequence (between -1 and 1)
Periodic (repeats, for every multiple of 2π in the input)

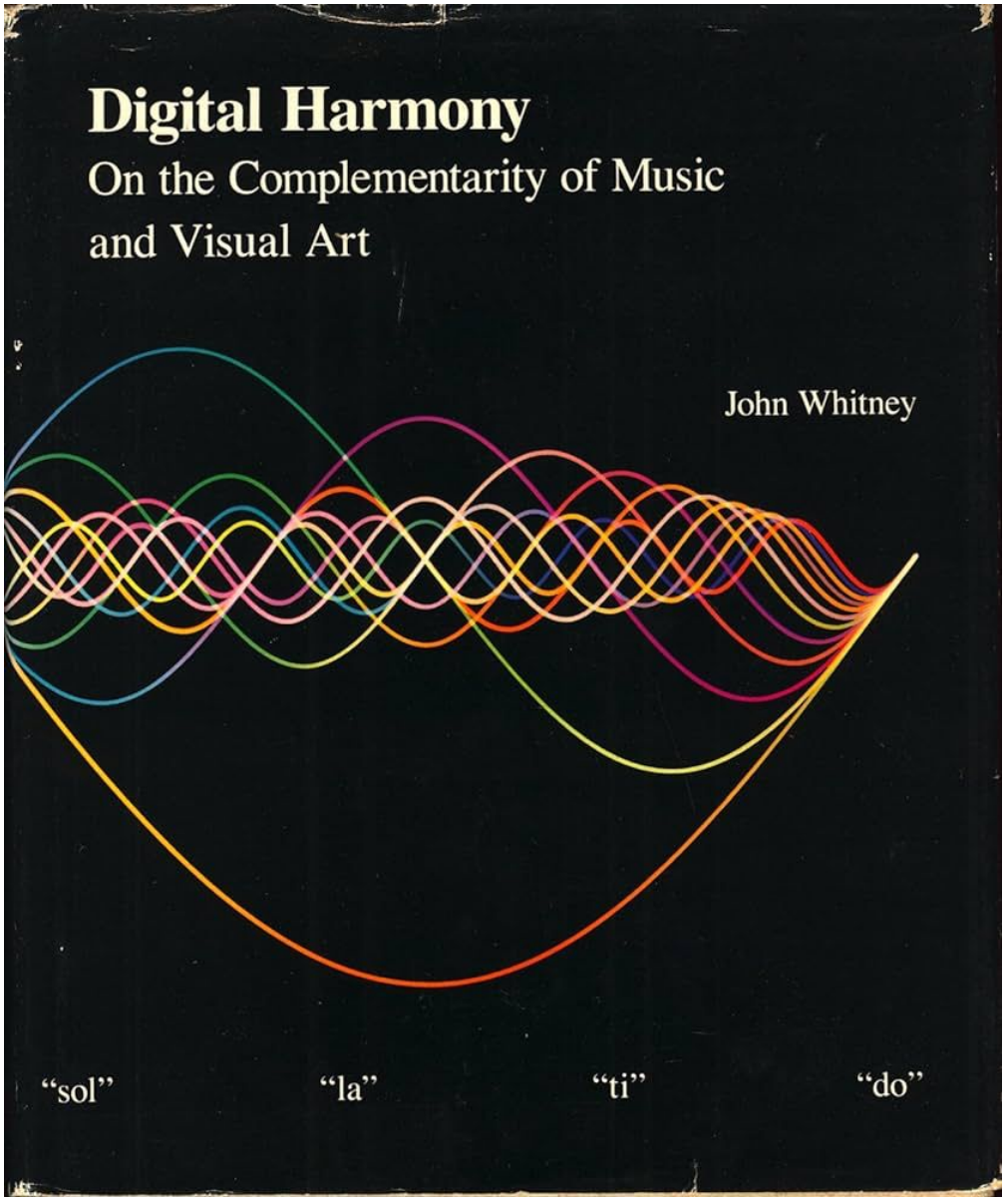
Sine and cosine waves

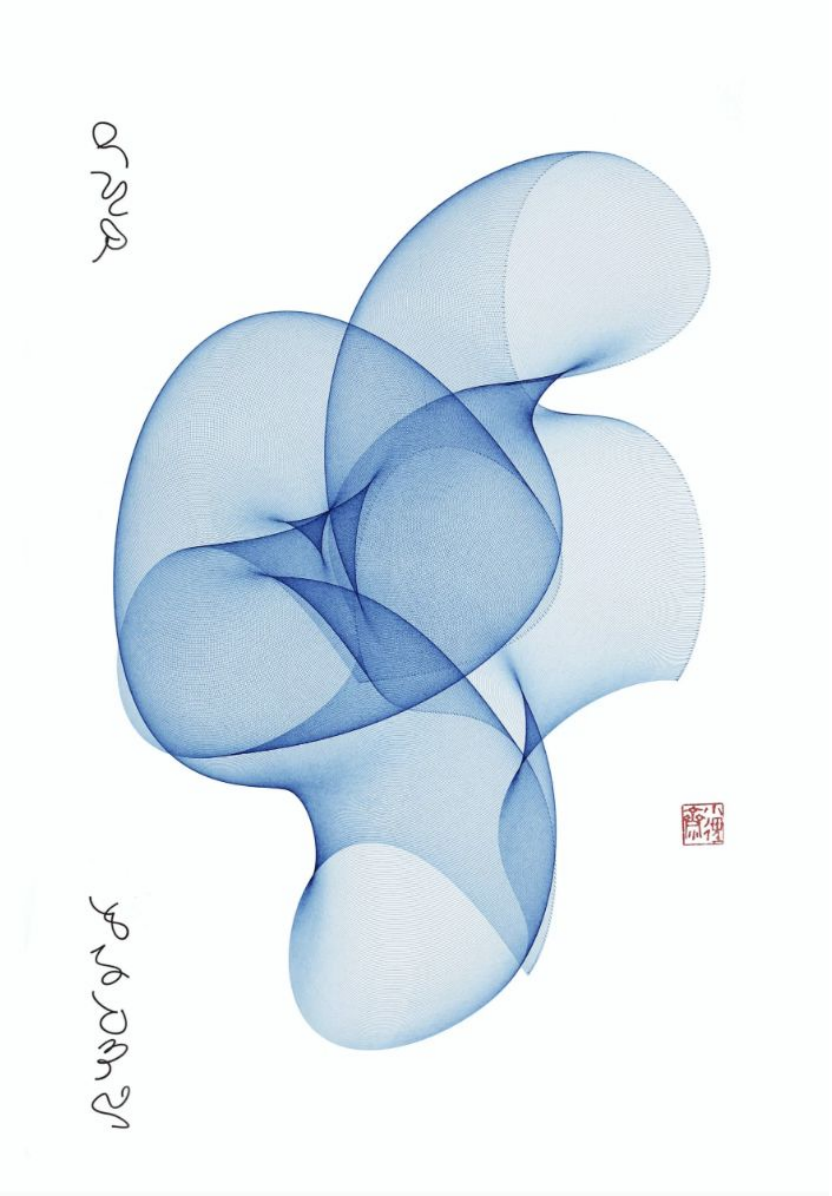





```
float wave = sin(t*frequency + phase)* amplitude;
```

sineWave





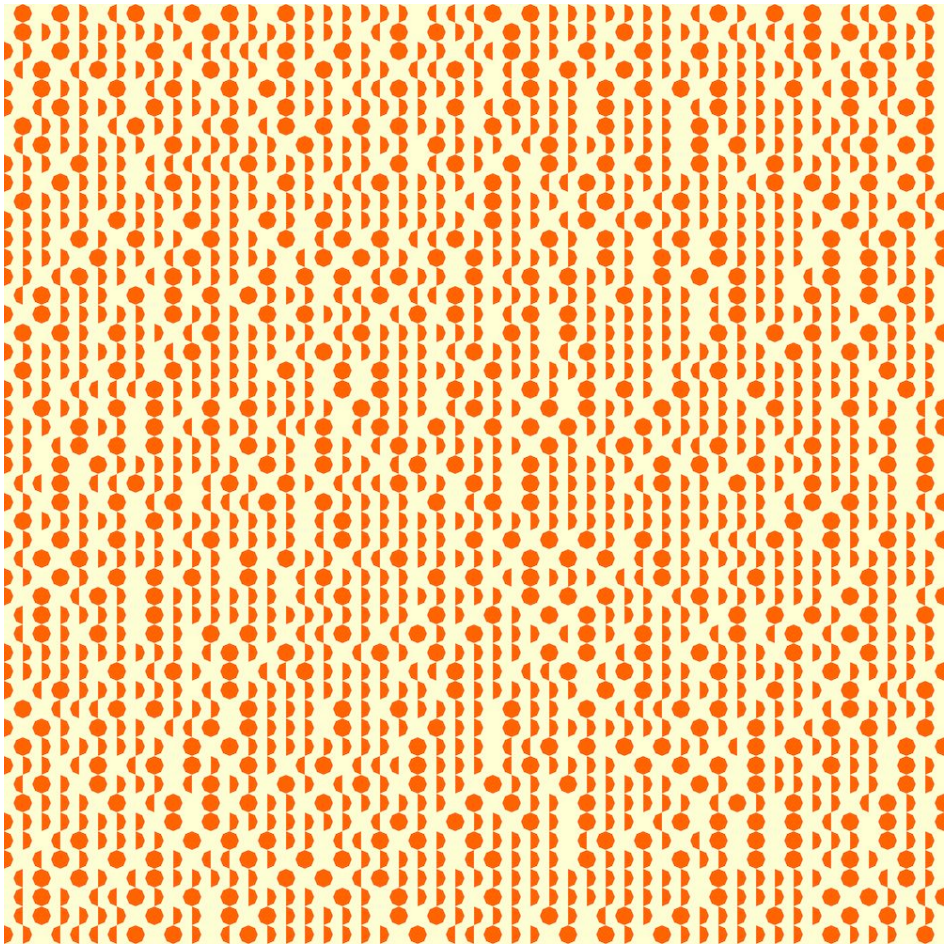
[John Witney]

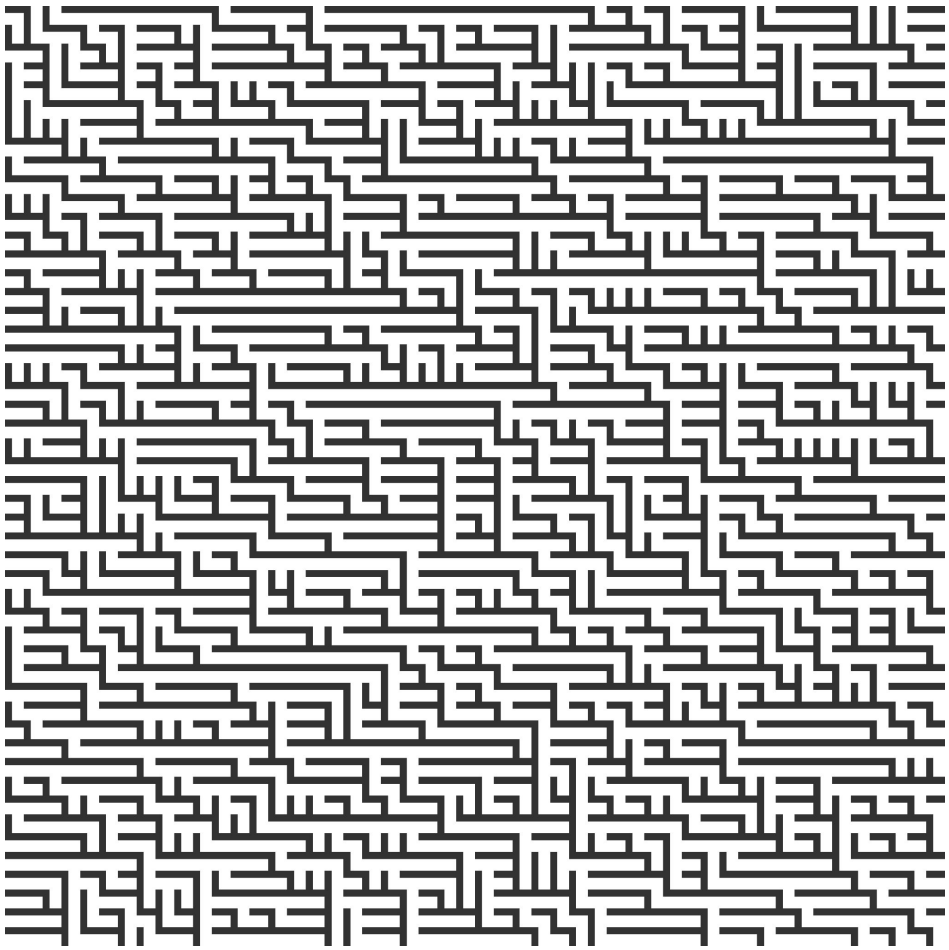
[Roman Verostko]

Break

Let’s use conditionals & loops together

to make images like these

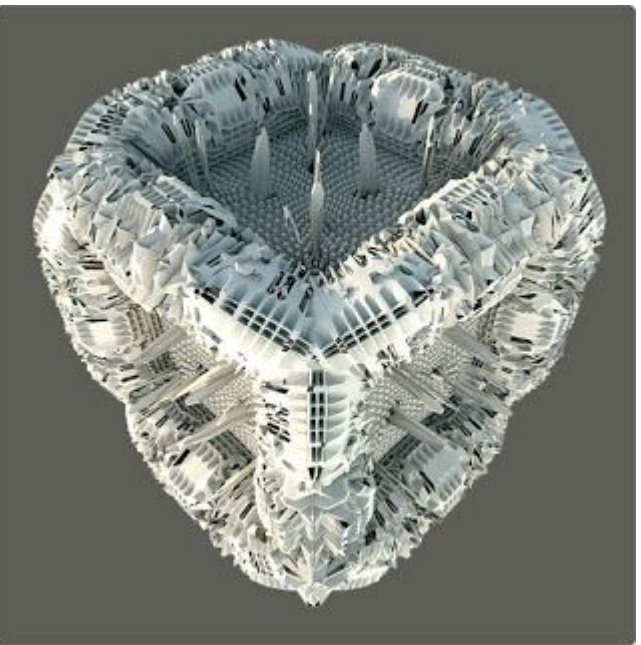


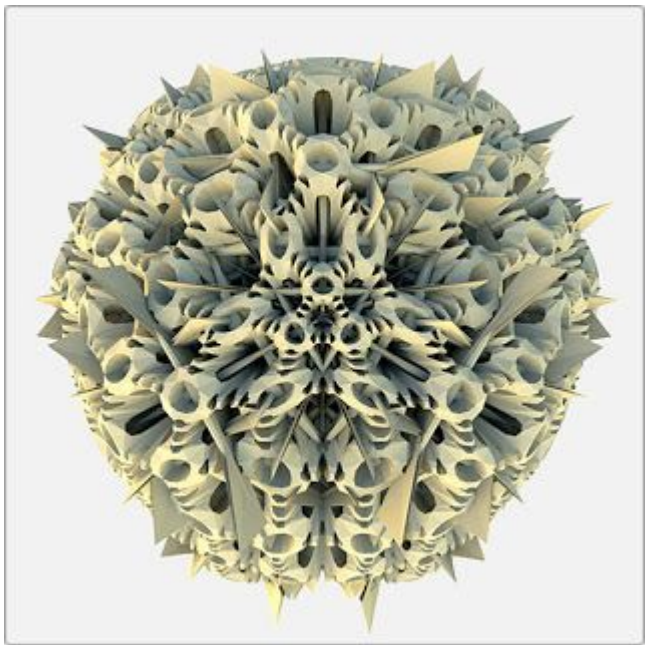


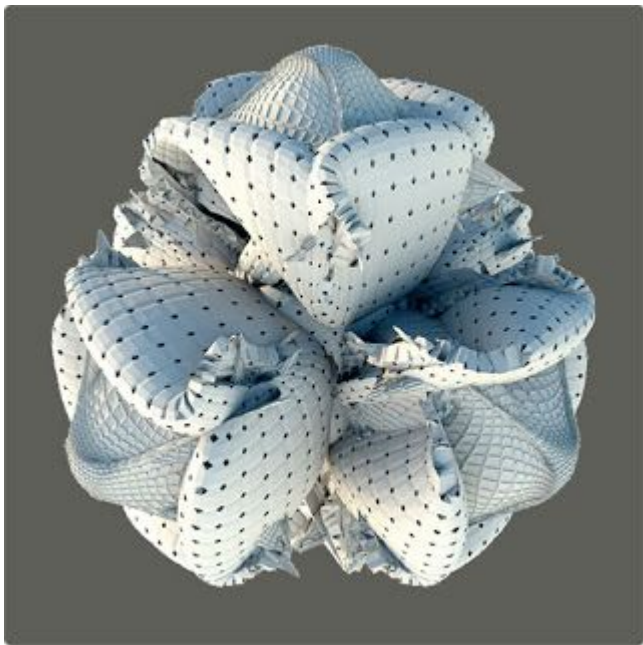
Generative Art

"Generative art refers to any art practice ***where the artist creates a process****, such as a set of natural language rules, a computer program, a machine, or other procedural invention, which is then set into motion* ***with some degree of autonomy contributing to or resulting in a completed work of art.****"*

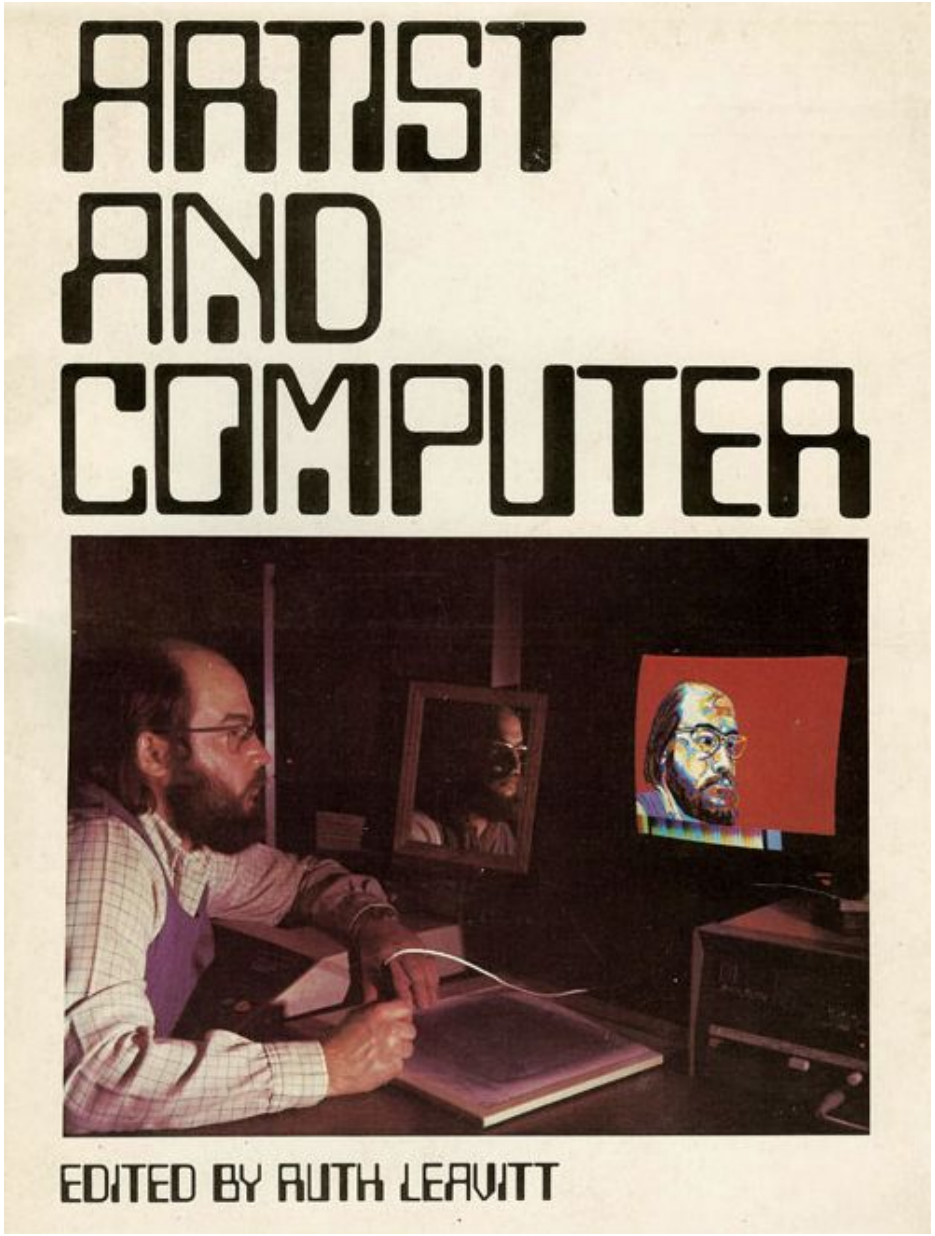
- Philip Galanter



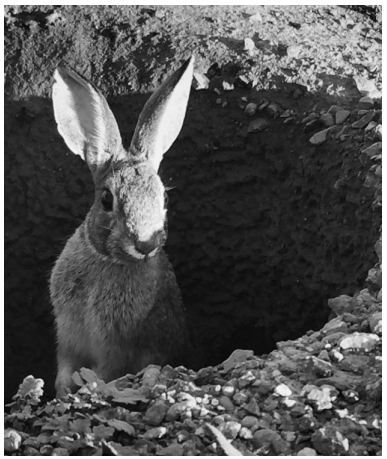




Early generative art

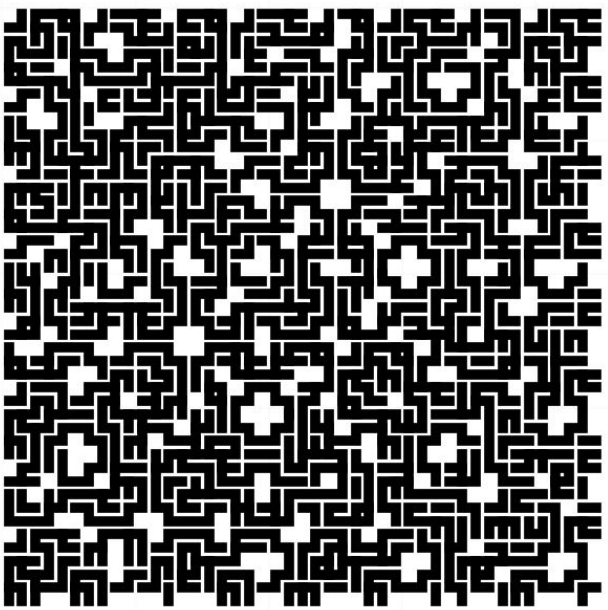


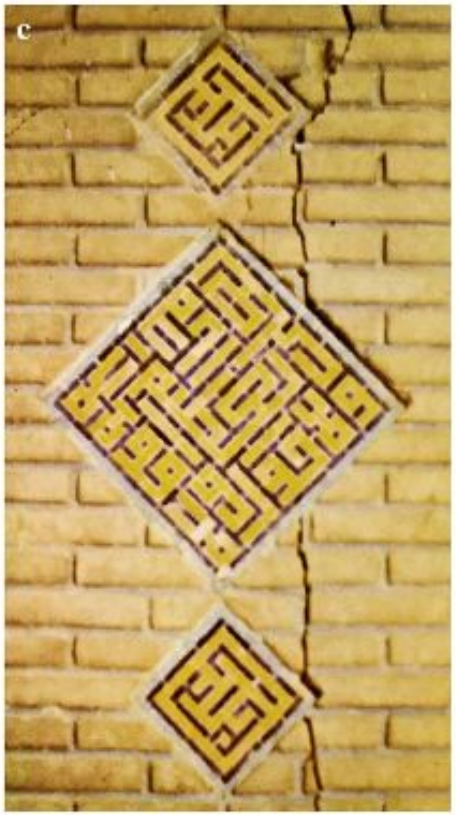
GREAT RESOURCE!
(inspiration and branching out to a rabbit hole)

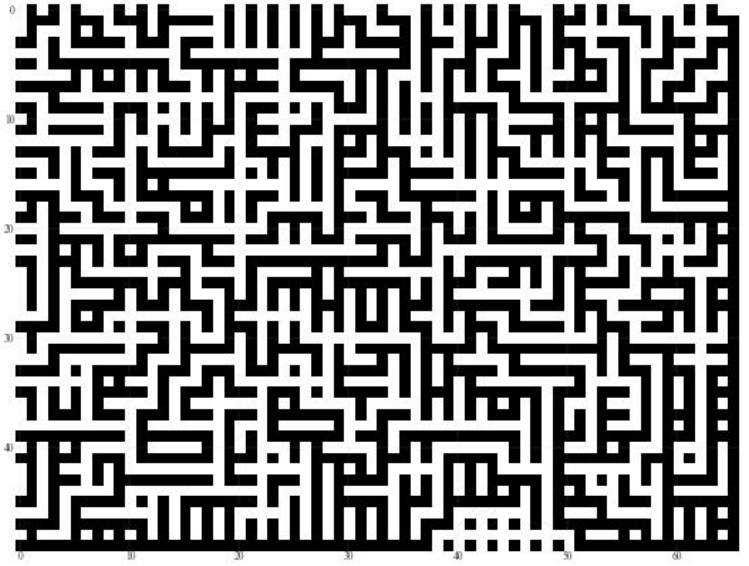


Square Kufic

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







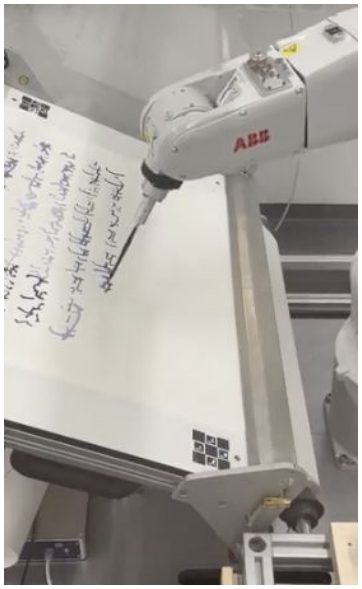
Sabetfard & Hadi Nadimi (2020) Generating Square Kufic Patterns Using Cellular Automata

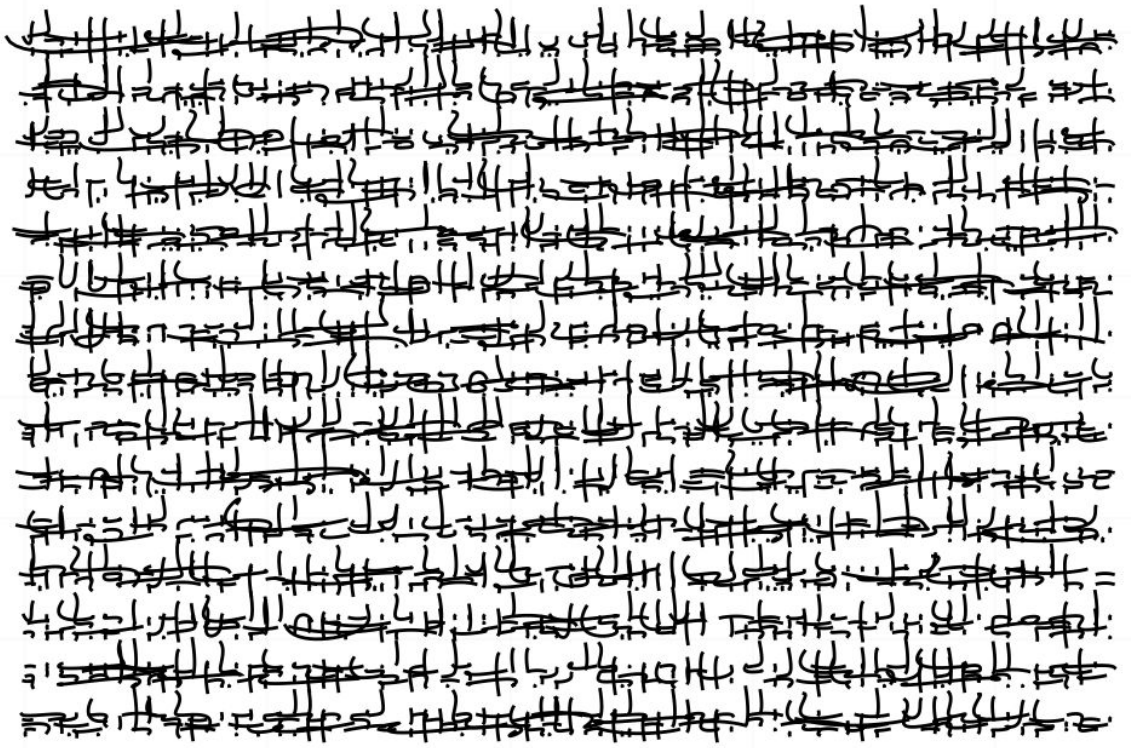
https://link.springer.com/article/10.1007/s00004-019-00454-3

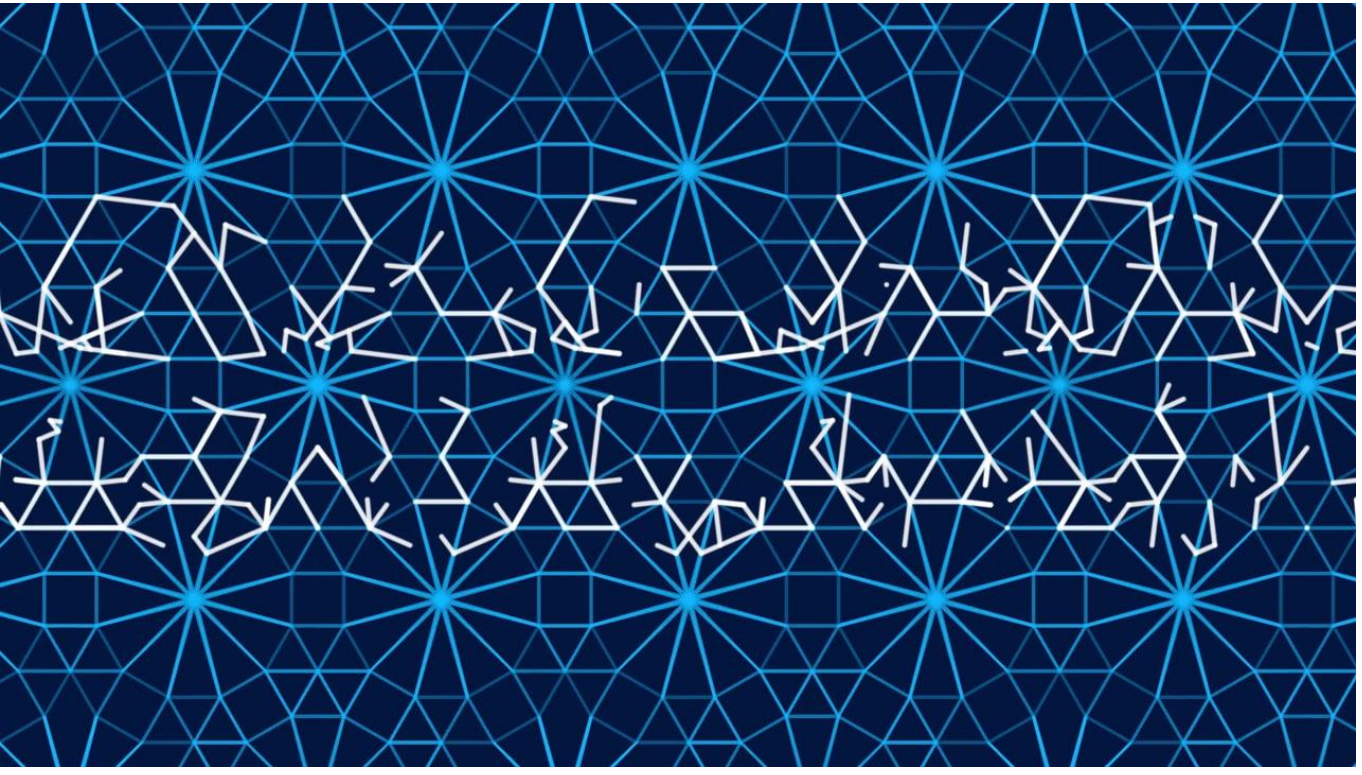
kufic

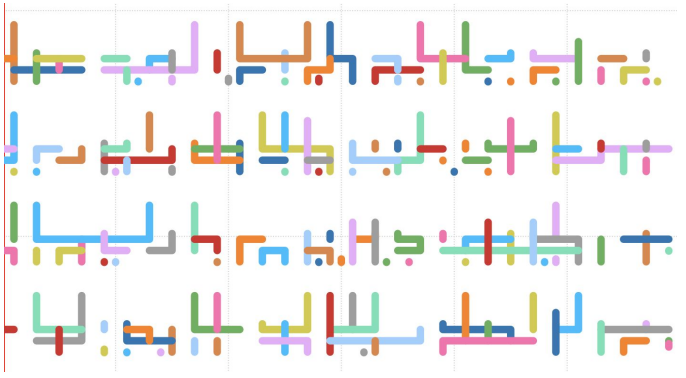
Continuum

Daniel Berio
Liat Grayver
Nora Al Badri









introducing random into our sketches

```
float num = ofRandom(5);
```

- returns a float number between 0 and 5 (but not 5)

```
float num = ofRandom(3,5)
```

- returns a float number between 3 and 5 (but not 5)

ofRandom gives a number from an "Uniform distribution":

Equal probability of picking any given number in the range

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

manipulating randomness

- Can use `ofRandom()` to choose actions with specific probabilities

```
float r = ofRandom(1); //generate between 0-1

if (r < 0.3) {
    //do something: 30% prob
}else if (r < 0.7) {

  //do something: 40% prob

}else {
  // do something: 30% prob

}
```

Random indices

```
int num = ofRandom(5);
```

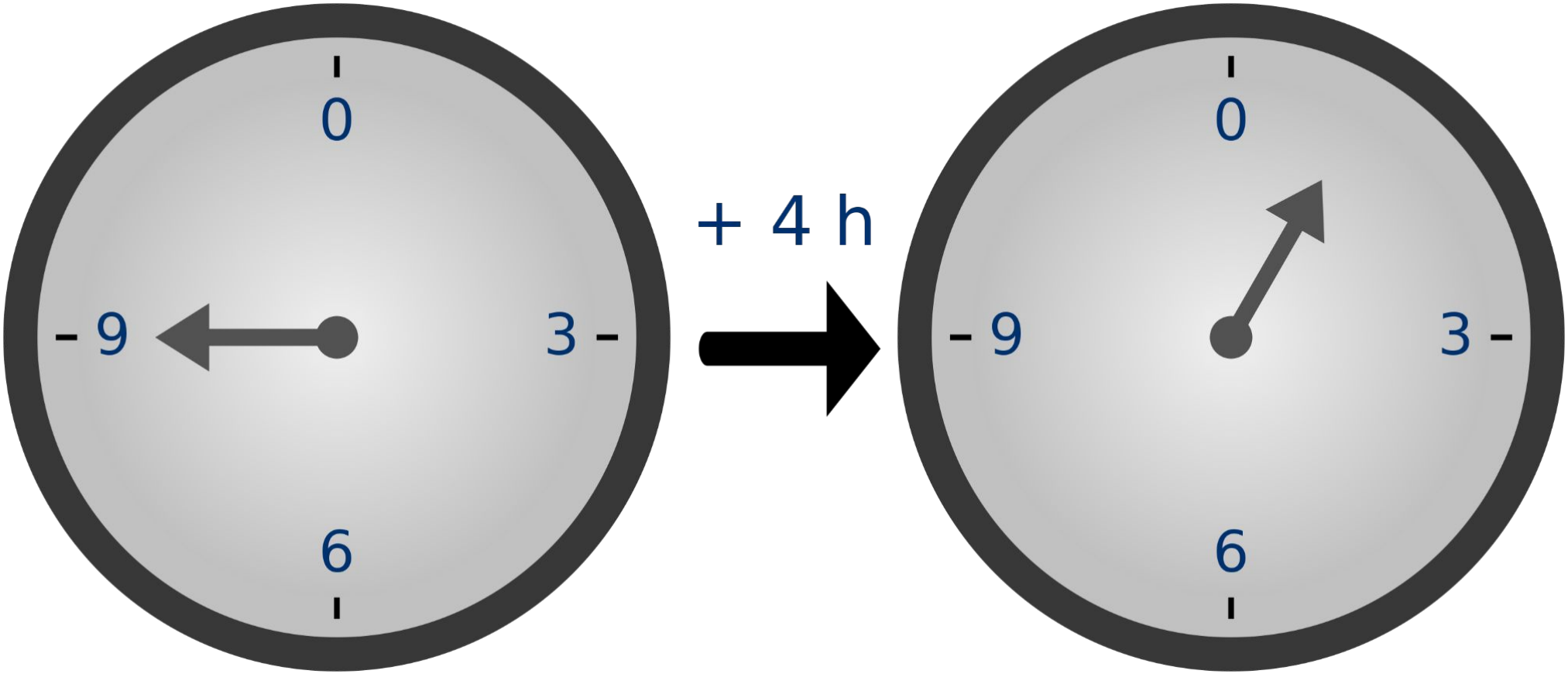
- Gives us an integer between 0 and 4 (included)

```
int num = ofRandom(3,5);
```

- returns an integer between 3 and 4 (included)

```
int num = rand()%3;
```

- The “C” way, like `ofRandom(3)`. Modulo operator

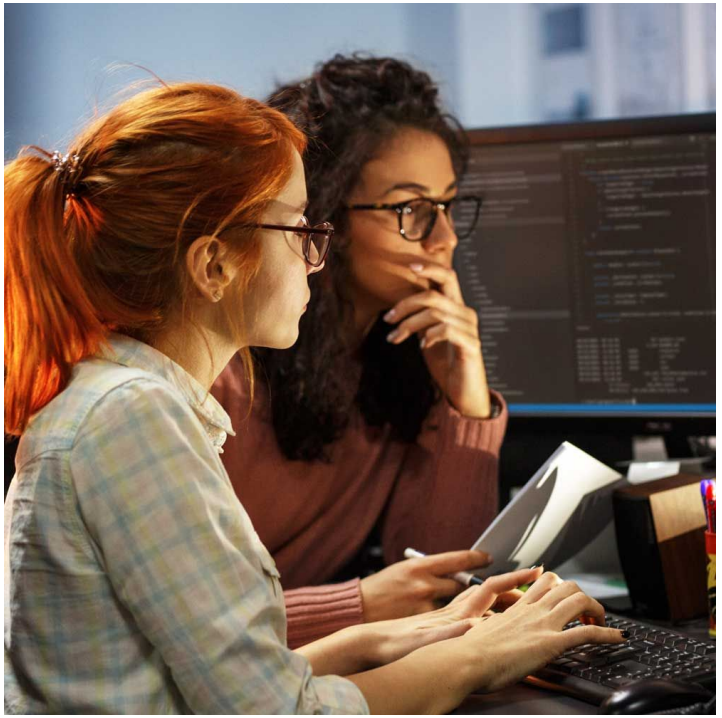


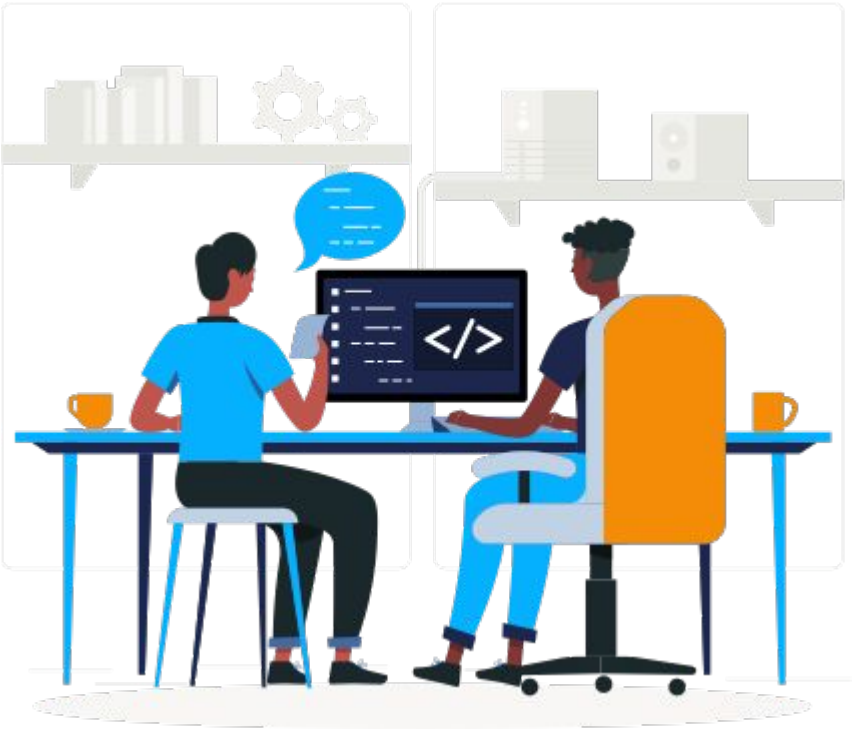

```
hour = (hour + 1)%12
```

Lab assignments

- Do the lab activities in the order provided! The texture lab activity is important for the homework
- Homework assignment:
  - Create a ***generative texture*** that uses ***repetition*** and ***randomness***.
  - Curate two or three ***variations*** that you like, showing how randomness changes the final ***output*** of your system.

Pair programming





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