introduction to programming in

“… computer programming is a process of self-humiliation. It is about finding out how poor your initial assumptions about your own abilities are, you feel frustration and you need to fight against your own attention deficit disorder.”

Julian Oliver

programming

- Setting parameters for future behavior.
- Setting rules for different conditions.
- Almost everything that adjusts to new conditions is based on some type of programming.
- It's a contract between two parties: the programmer and the compiler.

Bambi

PATONS 3-Ply BABY YARNS

Quantity .. .. **1 ball**
(1 ball will make 2 pairs of Bootees)

Length of foot .. .. 3½ ins.

Patons Beehive Knitting Needles, 1 pair No. 12, measured on a Beehive Needle Gauge.

Length of Ribbon.

ABBREVIATIONS: See page 24.

TENSION: 9 stitches to 1 inch in width, measured over plain smooth fabric. Check tension — see page 24.

Cast on 51 stitches.

1st and alt. rows (Wrong side of work).—Knit.

2nd row.—(Inc. once in next st., K. 23, inc. once in next st.) twice, K.1.

4th row.—(Inc. once in next st., K.25, inc. once in next st.) twice, K.1.

6th row.— (Inc. once in next st., K.27, inc. once in next st.) twice, K.1.

Continue inc. in this manner in every alt. row until there are 71 sts. on needle.

11th and alt. rows.—K.1, * P.1, K.3, rep. from * to last 2 sts., P.1, K.1.

12th row.—K.2, * P.3, K.1, rep. from * to last st., K.1.

14th row.—K.2, * P.1, w.r.n., P.2 tog., K.1, rep. from * to last st., K.1.

16th and 18th rows.—As 12th row.

20th row.—As 14th row.

22nd row.—Patt. 41, wool back, slip 1 knitways, K.1, p.s.s.o., turn.

23rd row.—Patt. 12, P.2 tog., turn.

24th row.—Patt. 12, wool back, slip 1 knitways, K.1, p.s.s.o., turn.

Rep. 23rd and 24th rows ten times, then 23rd row once.

46th row.—Patt. to end of row. (47 sts.)

47th row.—As 11th row.

48th row.—* K.1, wl. fwd., K.2 tog., rep. from * to last 2 sts., wl. fwd., K.2 tog.

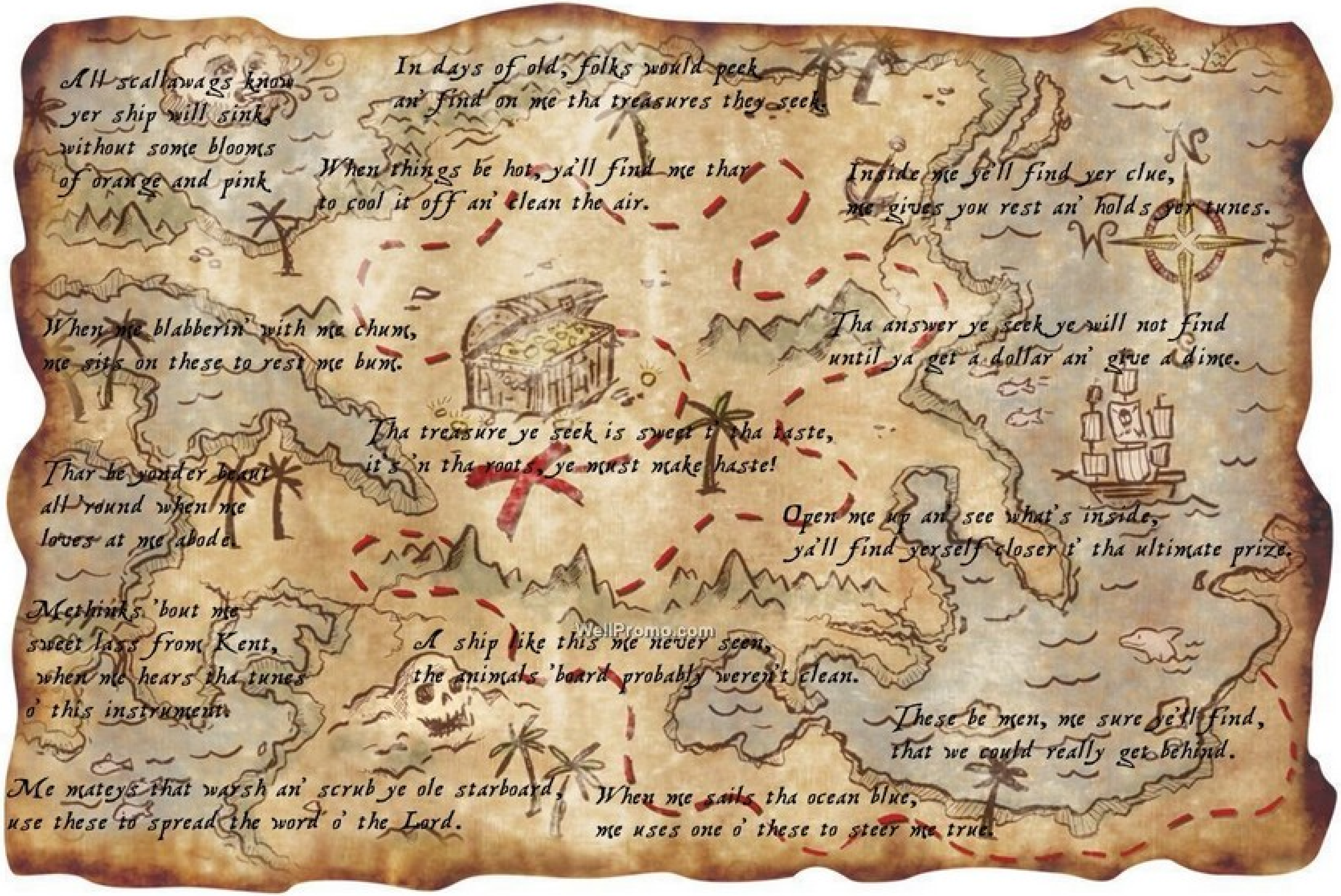
49th row.—Knit.

Rep. 49th row twenty-four times. Cast off.

Work another Bootee in same manner.

TO MAKE UP—With a slightly damp cloth and warm iron, press lightly. Using a flat seam, sew up leg and foot seams. Thread ribbon through holes at ankles. Fold over tops to form cuffs. Finally, press.

knitting instructions



All scallawags know
yer ship will sink,
without some blooms
of orange and pink

In days of old, folks would peek
an' find on me tha treasures they seek.

When things be hot, ya'll find me thar
to cool it off an' clean the air.

Inside me ye'll find yer clue,
me gives you rest an' holds yer tunes.

When me blabberin' with me chum,
me sits on these to rest me bum.

Tha answer ye seek ye will not find
until ya get a dollar an' give a dime.

Tha treasure ye seek is sweet t' tha taste,
it's 'n tha roots, ye must make haste!

Thar be yonder beast
all 'round when me
loves at me abode.

Open me up an' see what's inside,
ya'll find yerself closer t' tha ultimate prize.

Methinks 'bout me
sweet lass from Kent,
when me hears tha tunes
o' this instrument.

A ship like this me never seen,
the animals 'board probably weren't clean.

These be men, me sure ye'll find,
that we could really get behind.

Me mateys that warsh an' scrub ye ole starboard,
use these to spread the word o' the Lord.

When me sails tha ocean blue,
me uses one o' these to steer me true.

treasure map with directions

Why so many programming languages?

- Most are not suitable for all uses

- Many are designed for specific tasks:
  - Educational - LOGO, Basic.
  - Musical – Max, Supercollider
  - Network - javaScript, PHP
  - Scientific - Fortran, MatLab
  - Military, space - A++, ADA
  - Micro-chip - picBasic, Assembler

not all languages are of the same **level**

- Level = How close is the language to the architecture of the machine
- closer to architecture $\rightarrow$ lower level $\rightarrow$ foreign to the human eye
- Low level languages have a limited vocabulary.
- But, are faster to execute & demand less resources

like any human language it requires practice

machine language and assembly
(low level languages)

calculating the Fibonacci in machine language:

```
8B542408 83FA0077 06B80000 0000C383
FA027706 B8010000 00C353BB 01000000
B9010000 008D0419 83FA0376 078BD98B
C84AEBF1 5BC3
```

The only language that can run directly on the processor without modifications.
We would never write programs in this language.

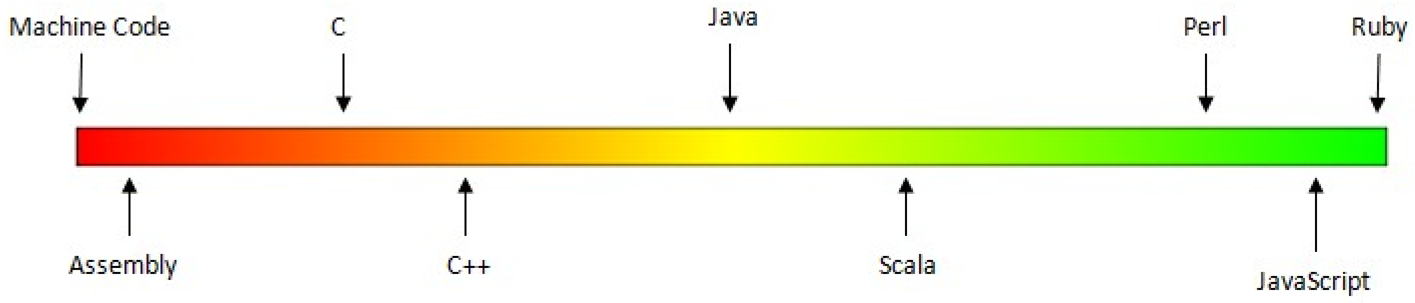
calculating the Fibonacci in Assembly:

```
fib:
  mov edx, [esp+8]
  cmp edx, 0
  ja @f
  mov eax, 0
  ret

  @@:
  cmp edx, 2
  ja @f
  mov eax, 1
  ret

  @@:
  push ebx
  mov ebx, 1
  mov ecx, 1

  @@:
    lea eax, [ebx+ecx]
    cmp edx, 3
    jbe @f
    mov ebx, ecx
    mov ecx, eax
    dec edx
  jmp @b

  @@:
  pop ebx
  ret
```

intermediate level languages for general use

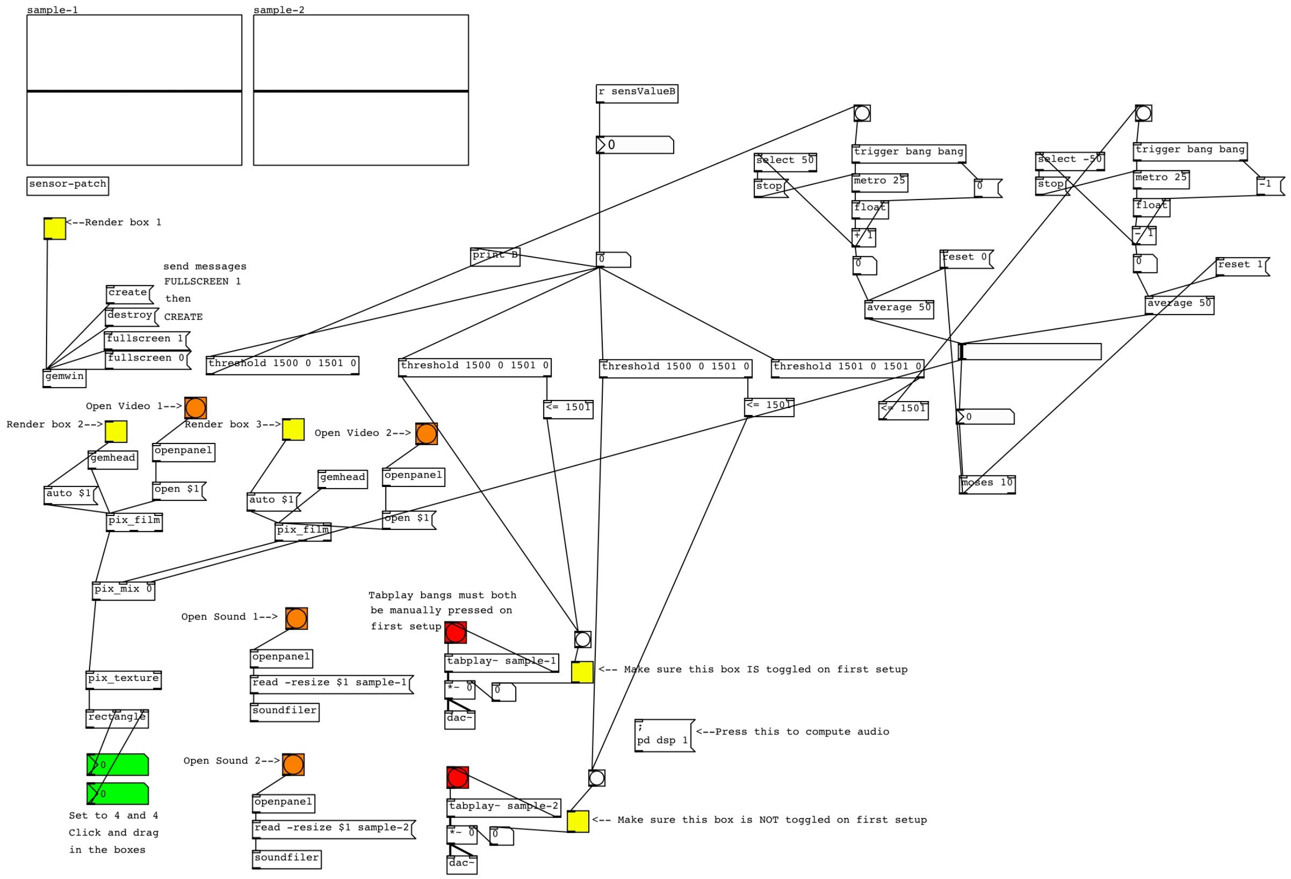
a vocabulary of about 30 terms that sound more familiar:

- while
- for
- if/ else”



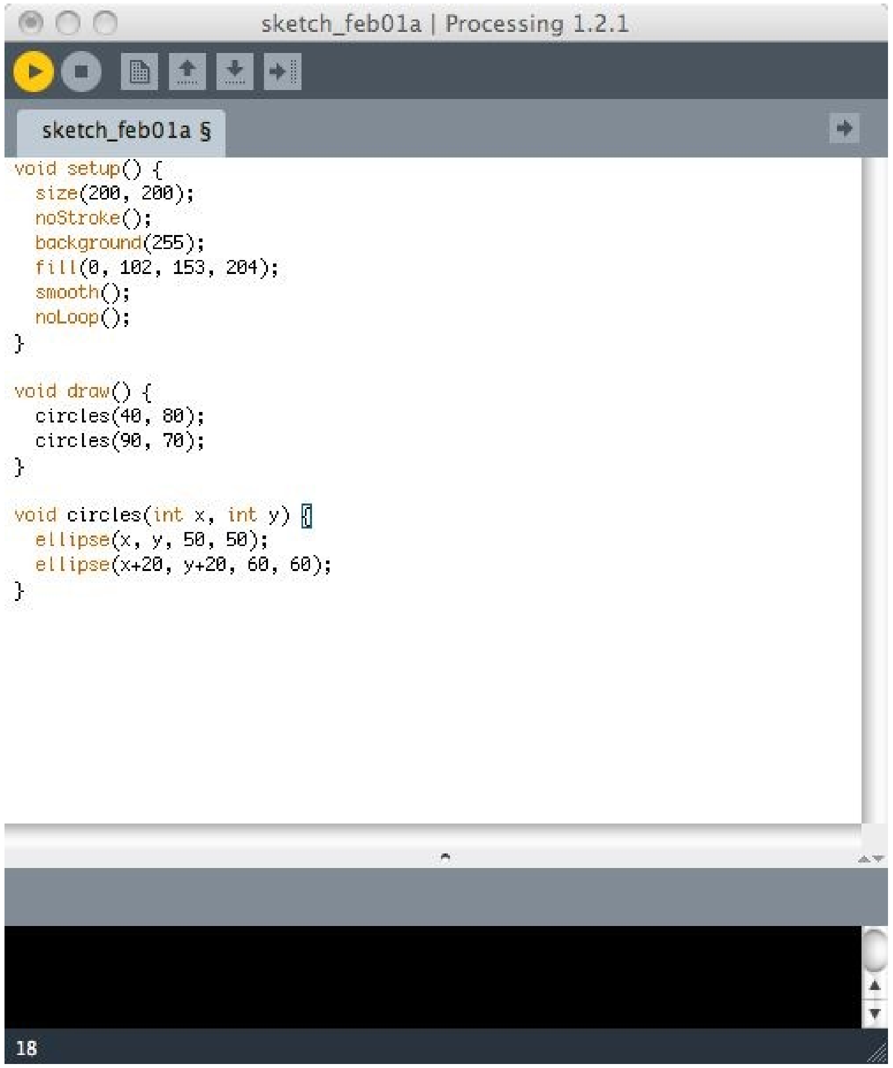
high-level languages

- Vocabulary with hundreds of words
- More abstract way of programming
  - processor registers vs. objects, variables, etc.
  - even visual programming possible
- Use of natural language elements
- But they use more memory and computer resources



Processing

IDE



```
void setup() {
  size(200, 200);
  noStroke();
  background(255);
  fill(0, 102, 153, 204);
  smooth();
  noLoop();
}

void draw() {
  circles(40, 80);
  circles(90, 70);
}

void circles(int x, int y) {
  ellipse(x, y, 50, 50);
  ellipse(x+20, y+20, 60, 60);
}
```

$$\begin{tikzcd}
\text{Processing} \arrow[d] \\
\text{Java} \arrow[d] \\
\text{Virtual Machine Code} \arrow[d] \\
\text{Operating System}
\end{tikzcd}$$

- easier to prototype in (simplified Java)
- slower to run on speedTestProcessing
- minimal IDE (can use Eclipse)
- memory management (garbage collection)
- can publish applets online
- can publish to Android and (with 3rd party tools) to iPhone

- a bit more tricky to prototype in (C++)
- more powerful and flexible because more low level
- faster execution speedTestOfx
- choice of IDE (XCode on OSX, Code::Blocks, VC++)
- you manage your own memory
- you publish your project as a native application
- supports iOS, OSX, Linux, Android, armv6, armv7 platforms

what is OF ?

- a toolkit for artists/designers working with interactive design and media art
- provides a simple and intuitive framework for experimentation
- works as a glue, wraping together several commonly used libraries:
  - OpenGL, GLEW, GLUT, libtess2 and cairo for graphics
  - rtAudio, PortAudio or FMOD and Kiss FFT for audio input, output and analysis
  - FreeType for fonts
  - FreeImage for image saving and loading
  - Quicktime and videoInput for video playback and grabbing
  - Poco for a variety of utilities
- supports five operating systems (Windows, OSX, Linux, iOS, Android)
- four IDEs (XCode, Code::Blocks, and Visual Studio).

OF FAQ

- Is it a program?
- Is it a programming language?
- Is it a library?
- What's a framework?

OF history

Zach Lieberman

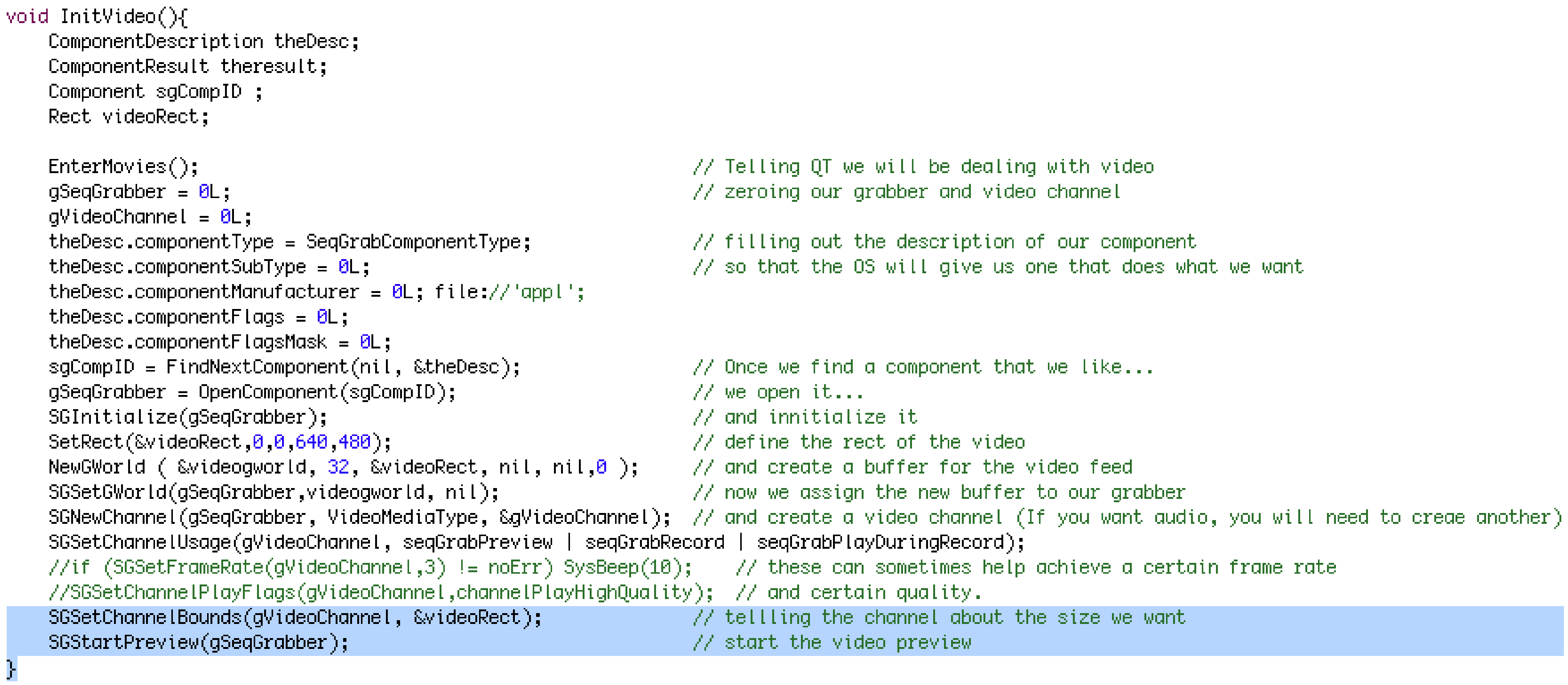
Arturo Castro

Theo Watson

Credit to:

- hundreds of other contributors
- Processing was great influence
- MIT Media Lab / Parsons School of Design

initializing the camera outside openFrameworks



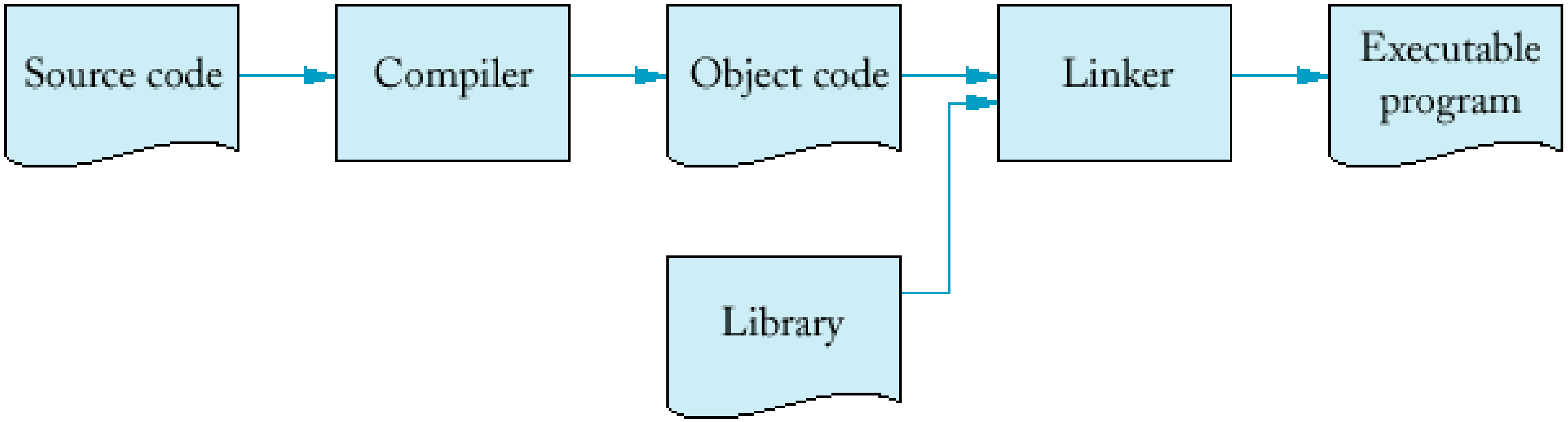
```
void InitVideo(){
    ComponentDescription theDesc;
    ComponentResult theresult;
    Component sgCompID ;
    Rect videoRect;

    EnterMovies();                                              // Telling QT we will be dealing with video
    gSeqGrabber = 0L;                                           // zeroing our grabber and video channel
    gVideoChannel = 0L;
    theDesc.componentType = SeqGrabComponentType;               // filling out the description of our component
    theDesc.componentSubType = 0L;                              // so that the OS will give us one that does what we want
    theDesc.componentManufacturer = 0L; file://'appl';
    theDesc.componentFlags = 0L;
    theDesc.componentFlagsMask = 0L;
    sgCompID = FindNextComponent(nil, &theDesc);                // Once we find a component that we like...
    gSeqGrabber = OpenComponent(sgCompID);                      // we open it...
    SGInitialize(gSeqGrabber);                                  // and innitialize it
    SetRect(&videoRect,0,0,640,480);                            // define the rect of the video
    NewGWorld ( &videogworld, 32, &videoRect, nil, nil,0 );     // and create a buffer for the video feed
    SGSetGWorld(gSeqGrabber,videogworld, nil);                  // now we assign the new buffer to our grabber
    SGNewChannel(gSeqGrabber, VideoMediaType, &gVideoChannel);  // and create a video channel (If you want audio, you will need to creae another)
    SGSetChannelUsage(gVideoChannel, seqGrabPreview | seqGrabRecord | seqGrabPlayDuringRecord);
    //if (SGSetFrameRate(gVideoChannel,3) != noErr) SysBeep(10);    // these can sometimes help achieve a certain frame rate
    //SGSetChannelPlayFlags(gVideoChannel,channelPlayHighQuality);  // and certain quality.
    SGSetChannelBounds(gVideoChannel, &videoRect);              // telling the channel about the size we want
    SGStartPreview(gSeqGrabber);                                // start the video preview
}
```

...while in ofx

```
ofVideoGrabber myGrabber;
myGrabber.initGrabber(640,480);
```

C++ compiling

- what is a compiler?
- hello world compilation hello world
- Xcode / code::Blocks - why even use an IDE?



OF + IDE

- directory structure
  - src
  - core addons + extra addons
  - examples
- project generator

  run by clicking on:

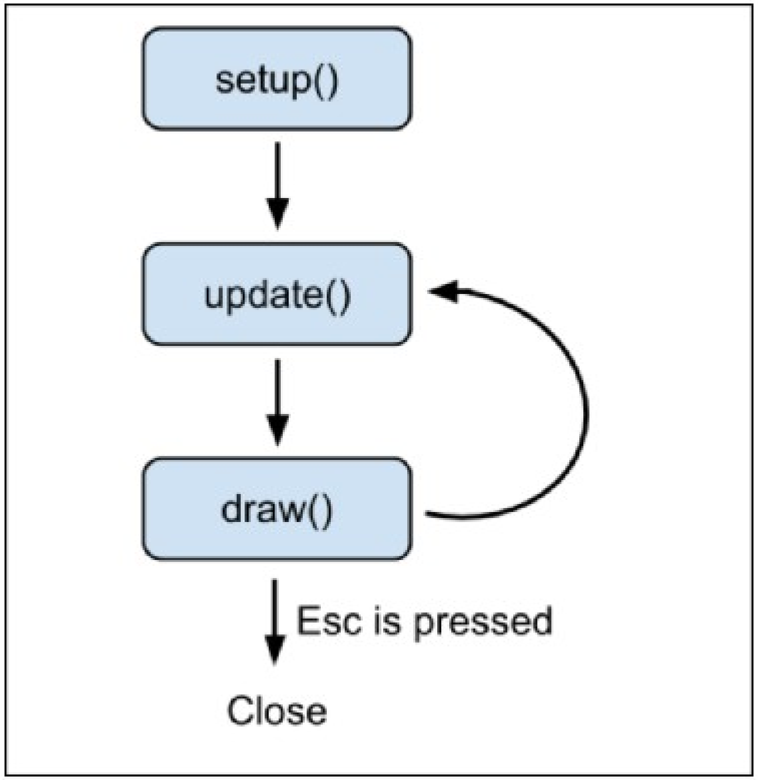
  openFrameworks/apps/projectGenerator/projectGeneratorSimple/bin/**projectGeneratorSimple**
- where your code should live

  *path to openFrameworks/***apps/myApps**
- resources

an empty project

- main.cpp
- ofApp.cpp
- ofApp.h
  - why multiple cpp files and header files?
    - **speeding up compile time**
    - **code more organized**: easier to find code
    - allows for separation of *interface* and *implementation*

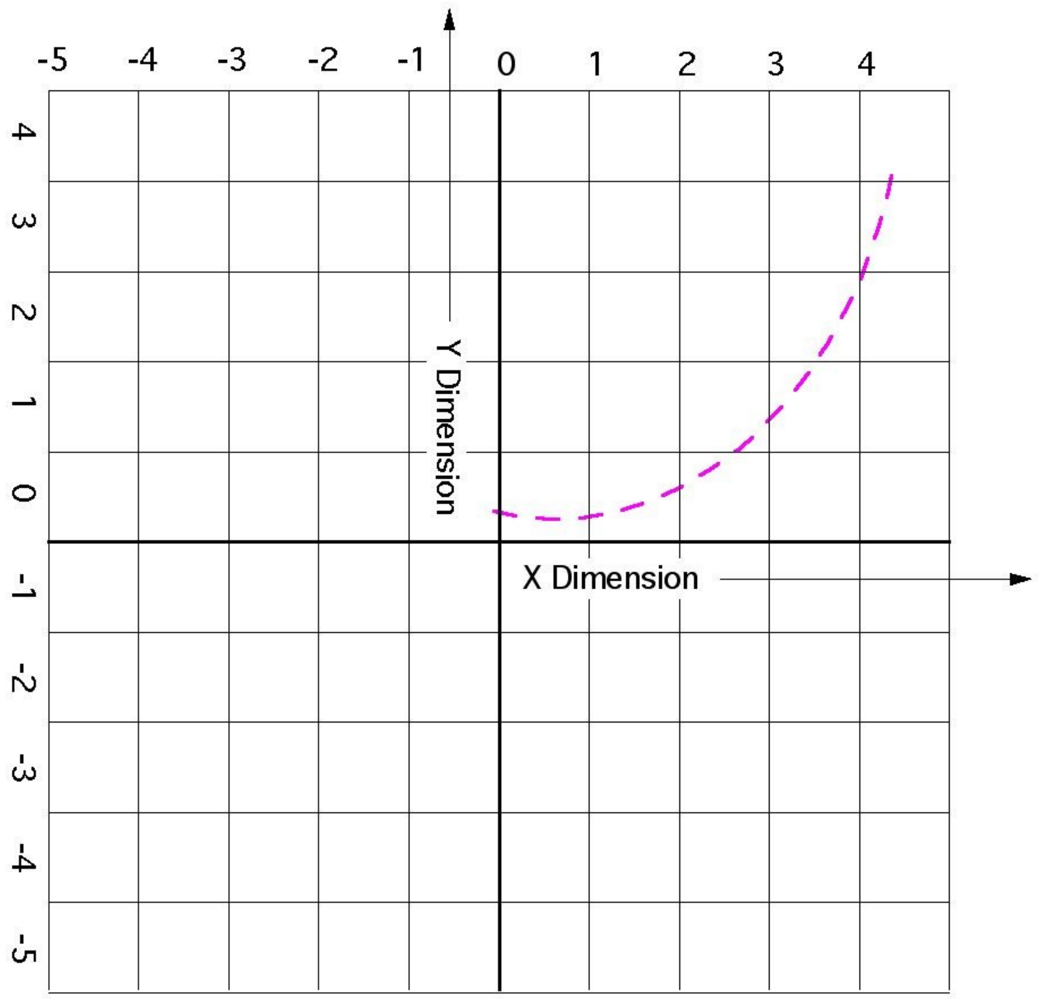
setup() vs. update() vs. draw()
& events



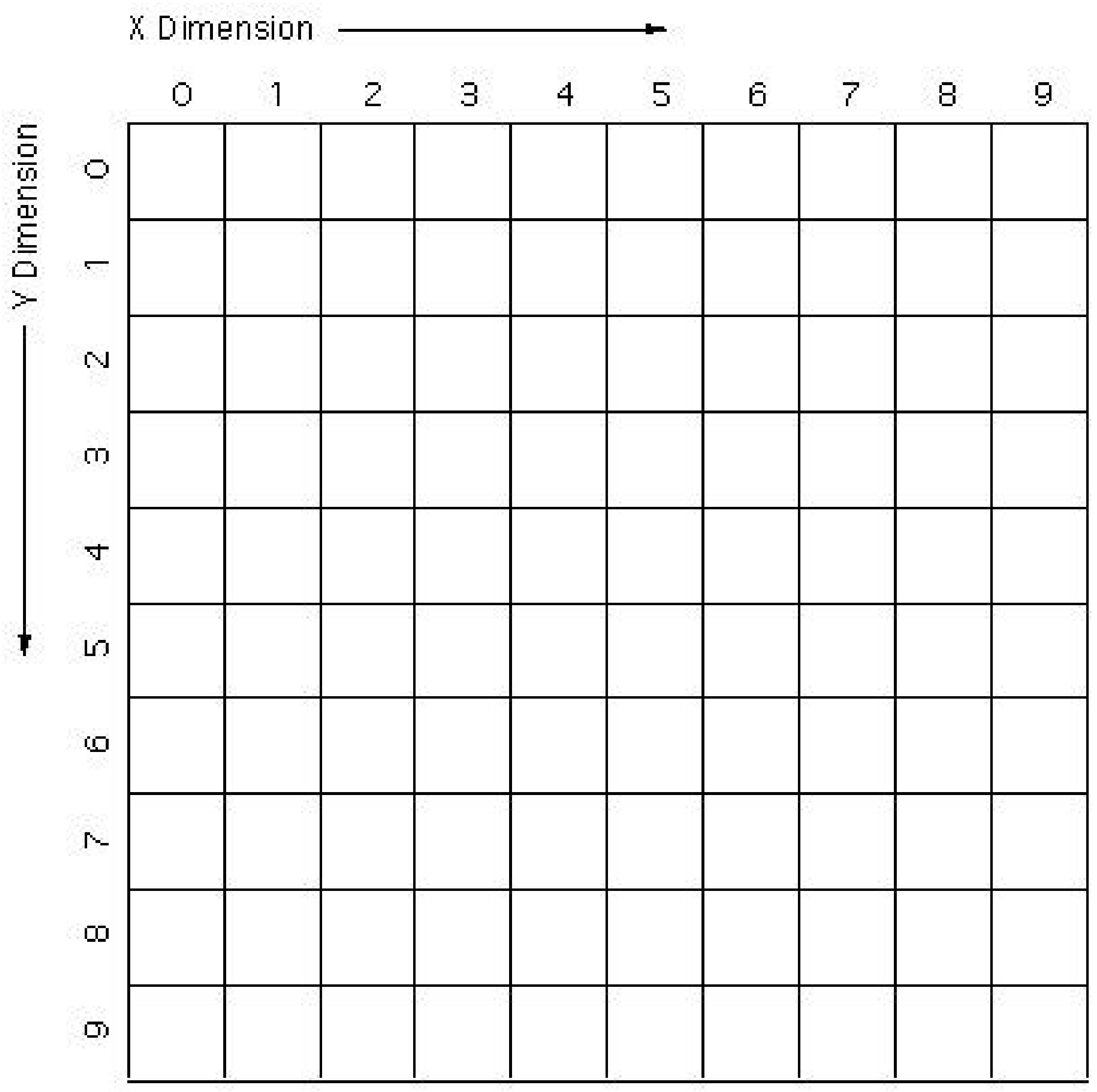
raed tihs snetecne*

* the brain can overcome errors in language, a machine can't

mathematical coordinate system



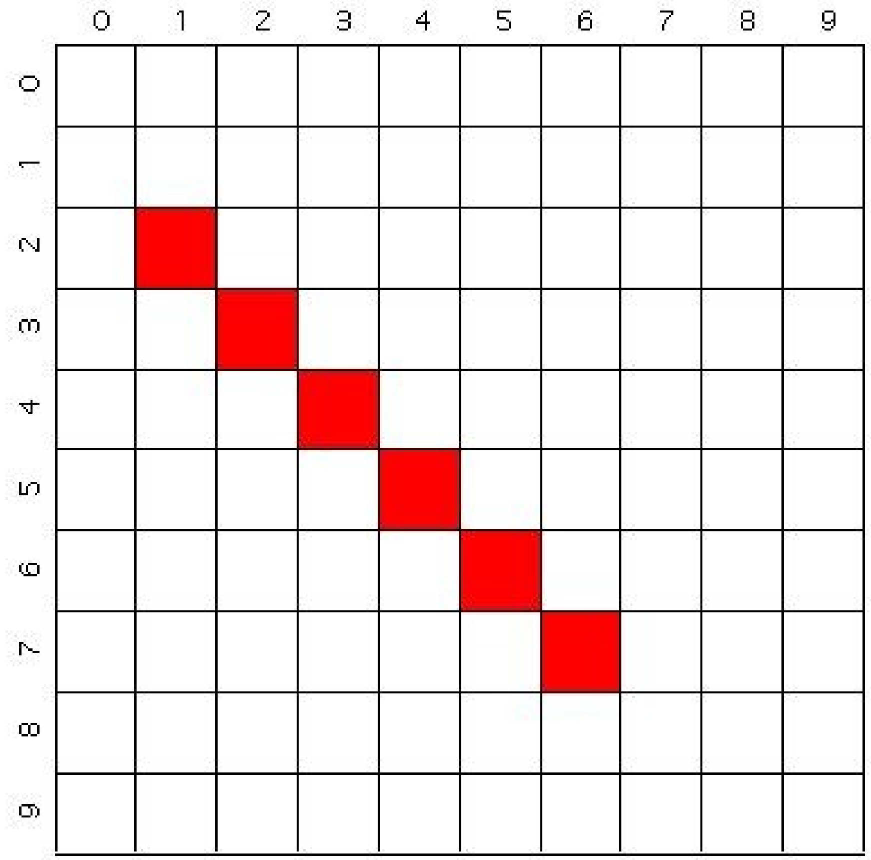
graphics coordinates



a line

```
ofLine(fromX, fromY, toX, toY);
```

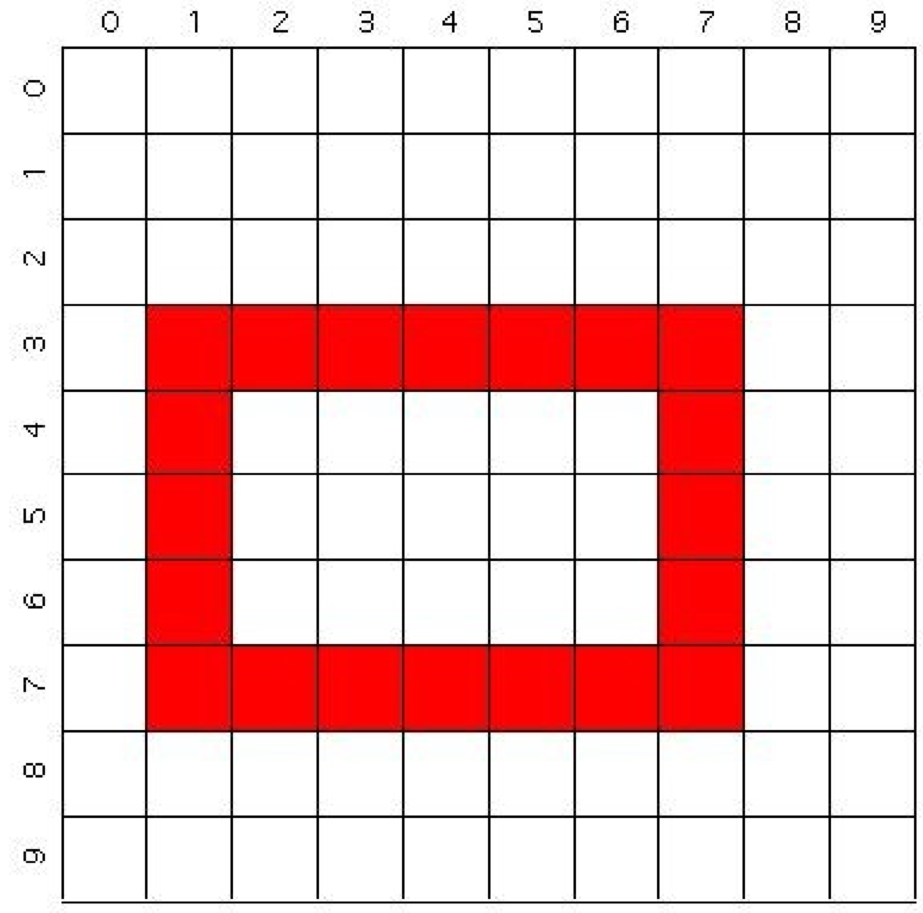
```
ofLine(1,2,6,7);
```



rectangle

- ofRect(topLeftX, topLeftY, width, height);

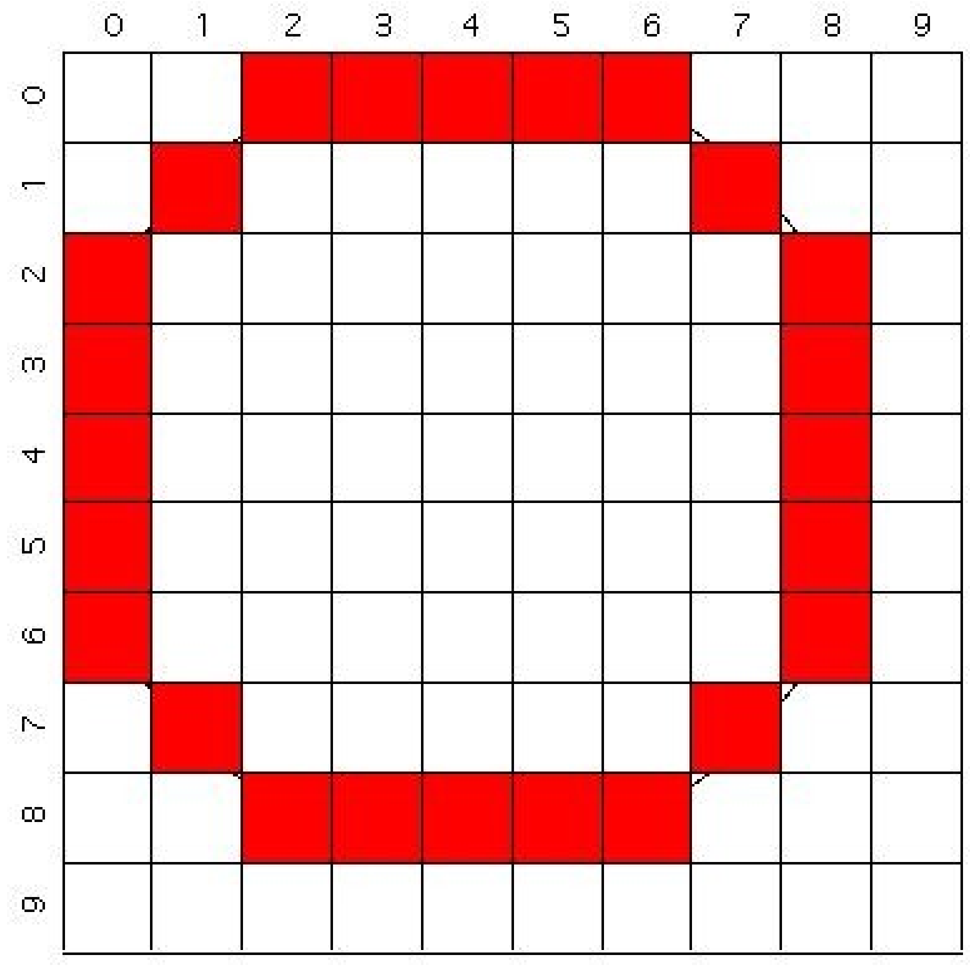
ofRect(1,3,7,5);



ellipse / circle

- ofEllipse(centerX,centerY,width,height);

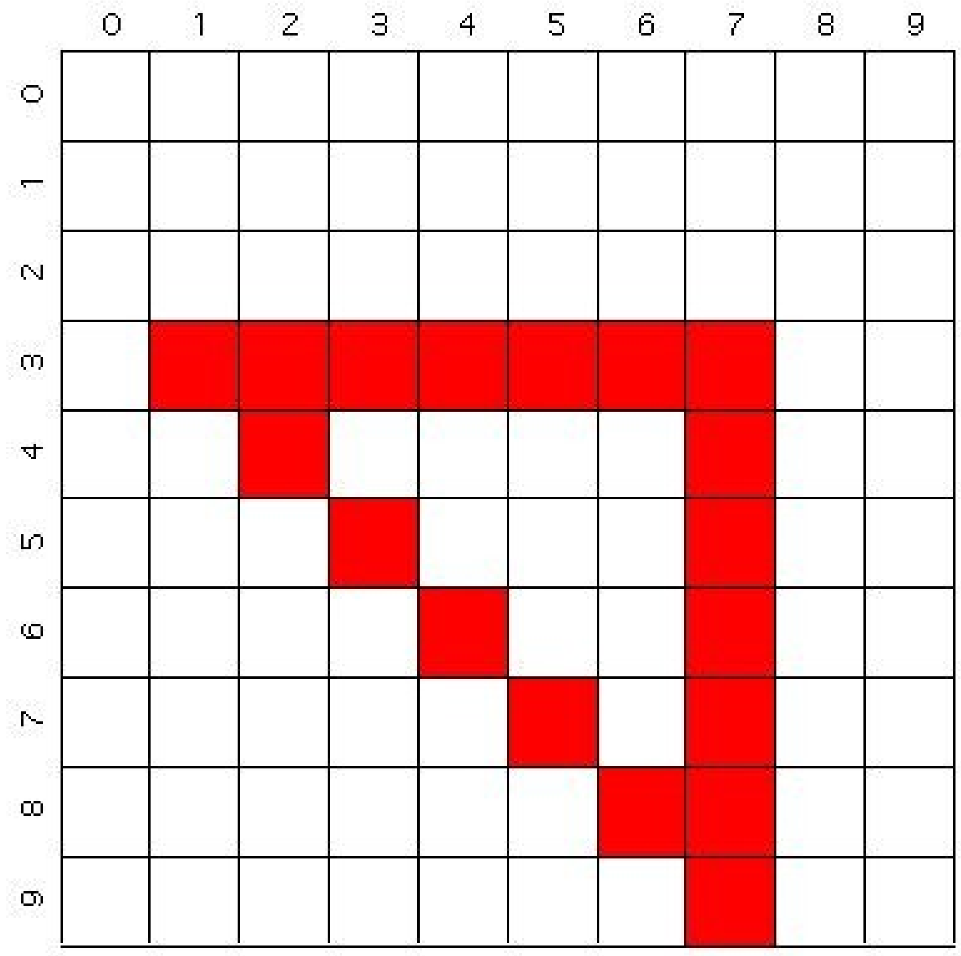
ofEllipse(4,4,9,9);



triangle

```
ofTriangle(point1X, point1Y, point2X, point2Y, point3X, point3Y);
```

```
ofTriangle(1,3,7,3,7,9);
```

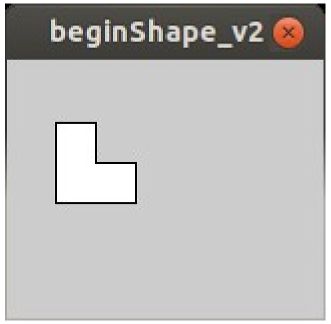


2D primitives

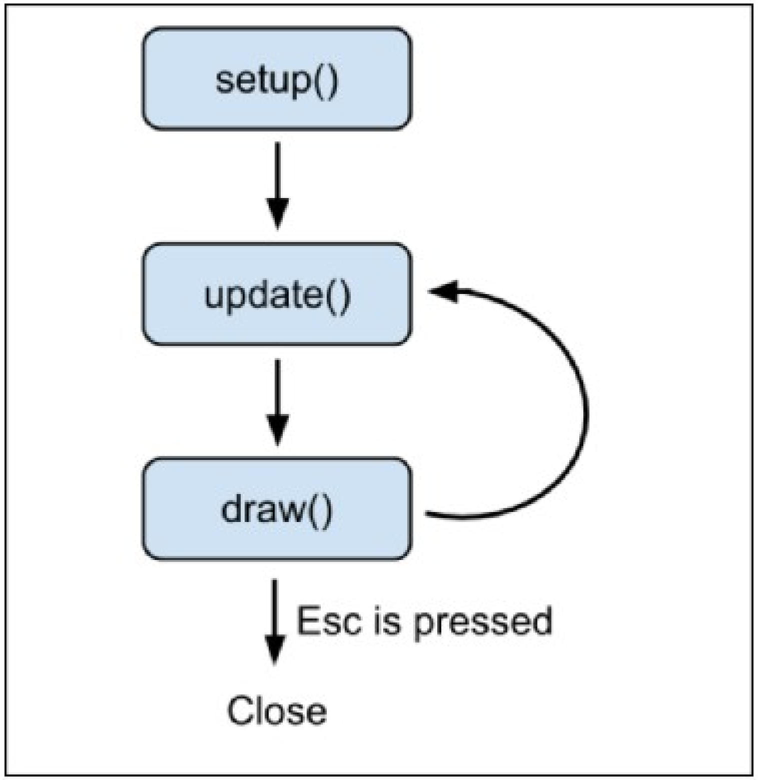
- ofLine(x1, y1, x2, y2);
- ofRect(x, y, w, h);
- ofTriangle(x1, y1, x2, y2, x3, y3);
- ofCircle(x, y, r);
- ofEllipse(x, y, w, h);
- ofCircle(x, y, r);

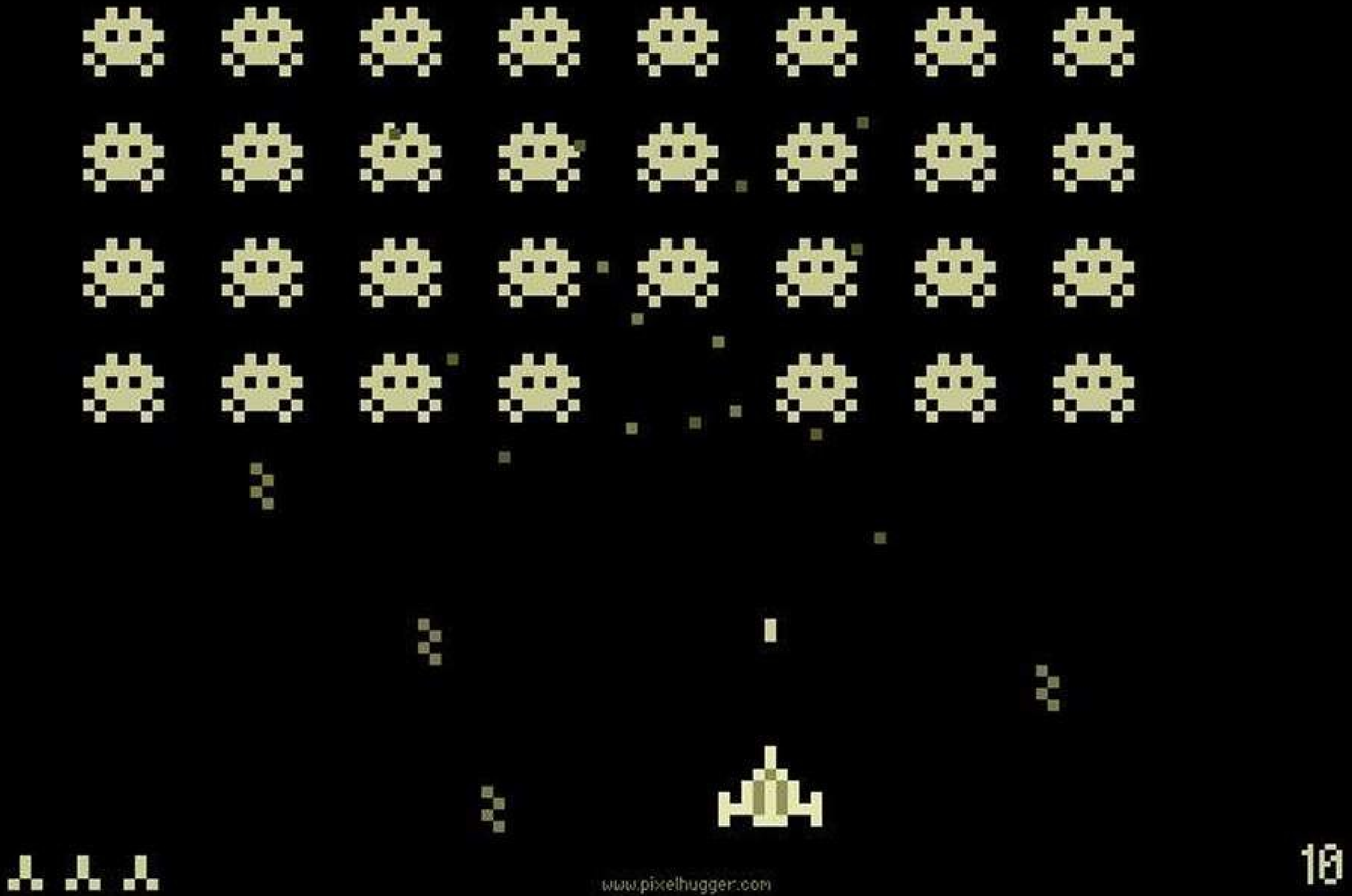
making your own shapes

```
ofBeginShape();
ofVertex(20, 20);
ofVertex(40, 20);
ofVertex(40, 40);
ofVertex(60, 40);
ofVertex(60, 60);
ofVertex(20, 60);
ofEndShape();
```



setup() vs. update() vs. draw() revisited!

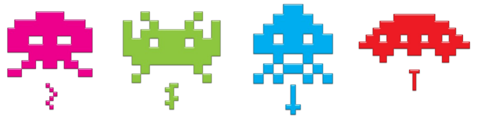



10
www.pixelhugger.com

`setup()` & `draw()`

- are found at the heart of almost every program

```
void ofApp::setup(){
   // code that we only want to run 1 time goes here
}
void ofApp::update(){
   // code that updates our variables goes here (more on this later)
}
void ofApp::draw(){
   // code that we want to be executed again and again goes here
}
```

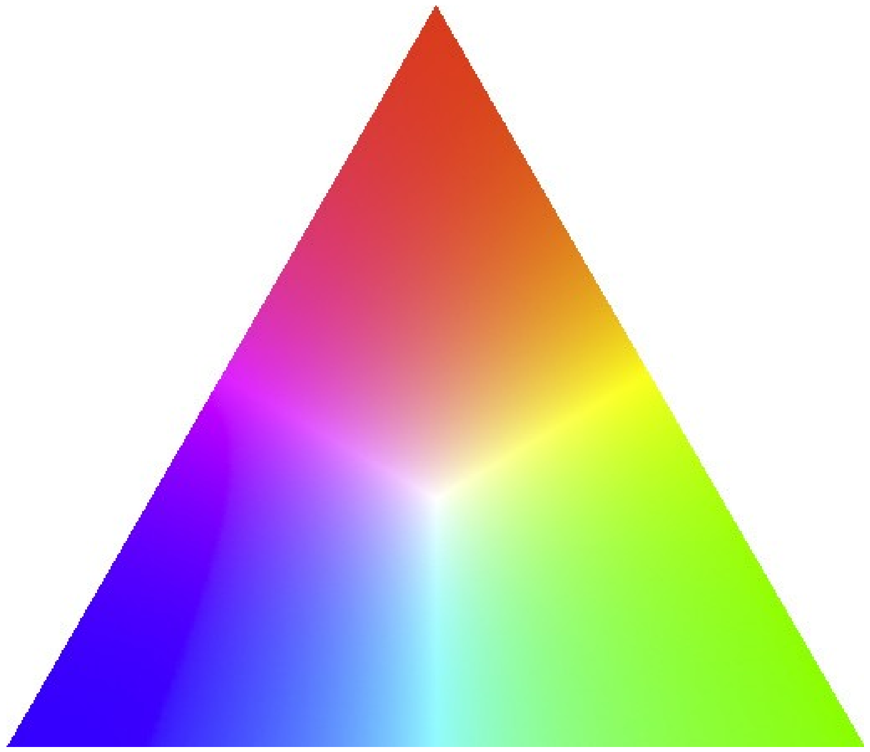
- this is how our software becomes dynamic
- example: Space invaders 
- where code is place affects when and how often it's going to be executed
  - 1 time: inside `setup()` - ideal for preparing our environment
  - many times: inside `draw()`

backgroundRefresh

when does oFx draw the elements of `draw();`

```
void ofApp::draw()
{
   background(255);
   rect(100,100,100,100);
   ellipse(100,100,50,50);
   triangle(100,100,50,50,20,20);
   //now !!
}
```

RGB color



some more commands

- ofBackground(R, G, B);
- ofSetBackgroundAuto(bool);
- ofSetColor(R, G, B, a);

- ofGetWidth(); / ofGetHeight();
- ofFill(); & ofNoFill(); shapeOutline
- ofSetLineWidth(aNumber);
- ofSetCircleResolution(aNumber);
- ofEnableAntiAliasing(); & disableAntiAliasing();
- mouseX / mouseY