programming for artists & designers

Daniel Berio

Session Summary

- Today’s goals topics overview
- More on Debugging
- New Coding Topics: (LOTS!)
  - Understanding openFrameworks docs
  - Variables
  - Scope
- Extending on drawing app
  - Some 2D transformations
  - ofMap
  - Conditionals

easier to prototype in (simplified Java)

slower to run on

minimal IDE

memory management (garbage collection)

can publish applets online

can publish to Android & (with 3rd party tools) to iPhone with a lot of pain!


speedTestProcessing

slightly slower to prototype in (C++)

faster, powerful & flexible because more low level

choice of IDE (XCode on OSX, QT Creator, VC++)

you manage your own memory (if using pointers)

you publish your project as a native application

supports iOS, OSX, Linux, Android, armv6, armv7 platforms


speedTestOfx

p5.js

easier to prototype (Javascript)

slower running in the browser

use a text editor (Brackets, Sublime, Atom etc)

memory management (garbage collection)

can be hosted anywhere online

lives in the ecology of the web, runs in browsers.

p5.js speedTestP5

what is OF ?

- works as a glue, wraping together several commonly used libraries, e.g.:
  - 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

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);          // tellling the channel about the size we want
    SGStartPreview(gSeqGrabber);                            // start the video preview
}
```

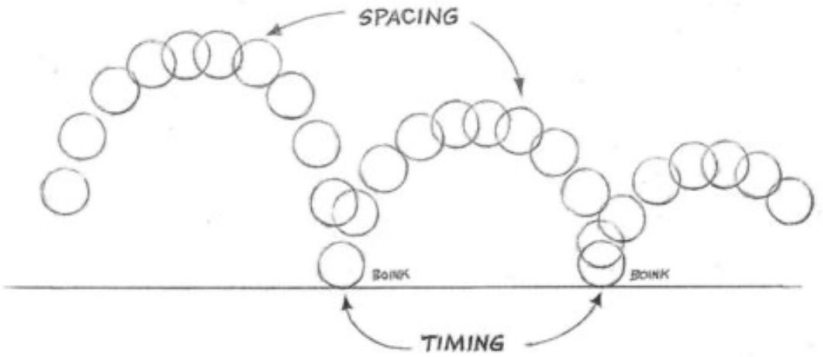
...while within openFrameworks

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

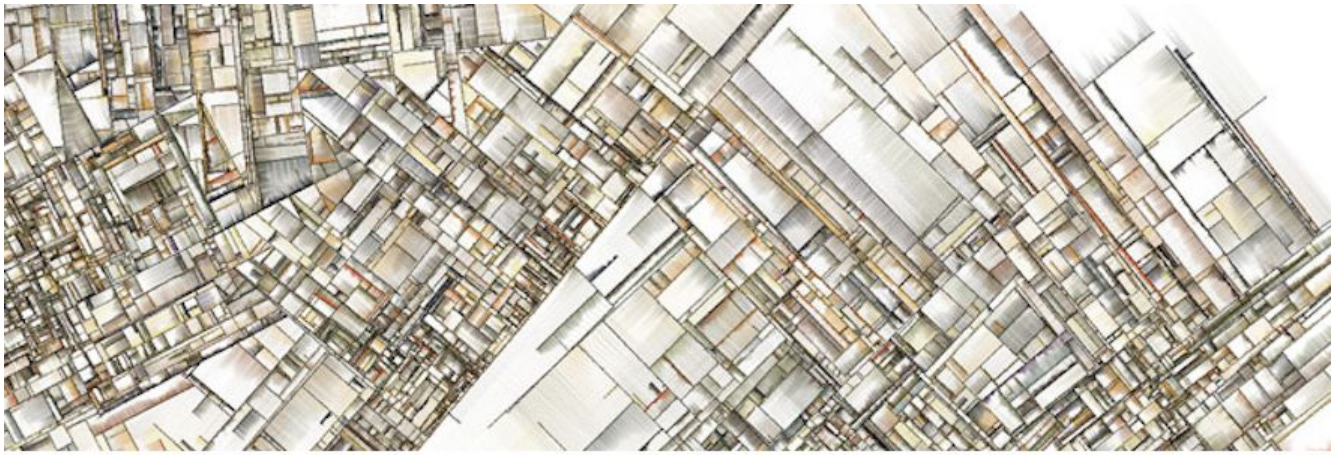
an empty project, what is an ofApp?

- 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***

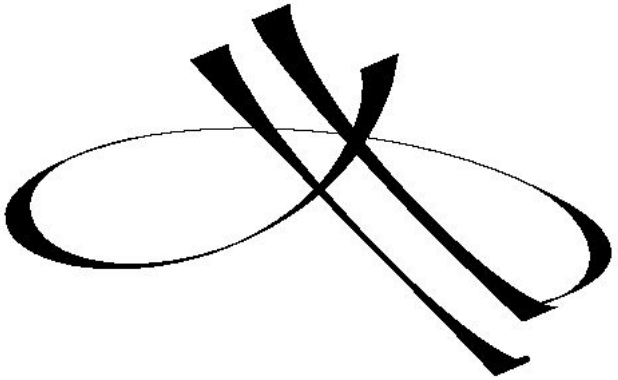
Our focus today



- Today we will be **drawing** with OF
- Using a “trick” that allows us to do quite complex drawings with very simple tools
  - Simply don’t clear the window!
  - Widely used in many Processing examples
  - Similar to “onion skinning” in animation (we show all frames)
- Unfortunately, there is a bug in OF, which requires us to setup the program in a particular way :(



Jared Tarbell:
http://www.complexification.net/gallery/machines/substrate/



Paul Haeberli
http://www.graficaobscura.com/dyna/

P5 version https://editor.p5js.org/golan/sketches/cZPRgx6q9

Basic drawing with OF

- You will use this template to start with today
  - Download, move to myApps, and good idea to duplicate it
  - Give the directory a name, e.g. myFirstDrawingApp
  - Use projectGenerator to create IDE files (demo, but see this)
- Basically, the animation example from yesterday
  - Using `ofDrawCircle(mouseX, mouseY, …)`
- But contains the following line in setup
  - `ofSetBackgroundAuto(false);`
  - This tells the app you don't want to to clear the background each frame

Console: easy way to debug

```
cout << "Hello World" << endl;
```

Helpful for ***debugging***

Debugging means finding & solving problems in code

Commenting out code

```
// this is a comment (for info)
// below a line of code is commented out
//ofDrawCircle(100,100,50);
// Can be useful to disable some
// functionality to better understand
// a problem
```

Helpful for ***debugging***

Debuggers

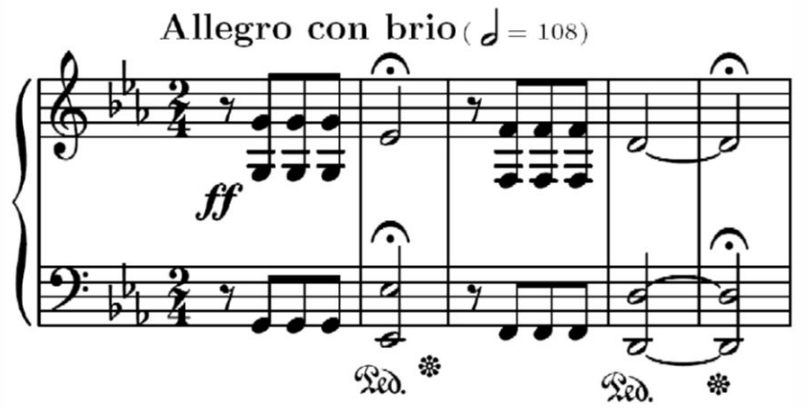


Figure 1.1 from [Müller, FMP, Springer 2015]

- Think of code as a “score”.
  - Debugger lets you step in the score note by note
- My recommendation is to just be aware of this for now, but to learn about it later.

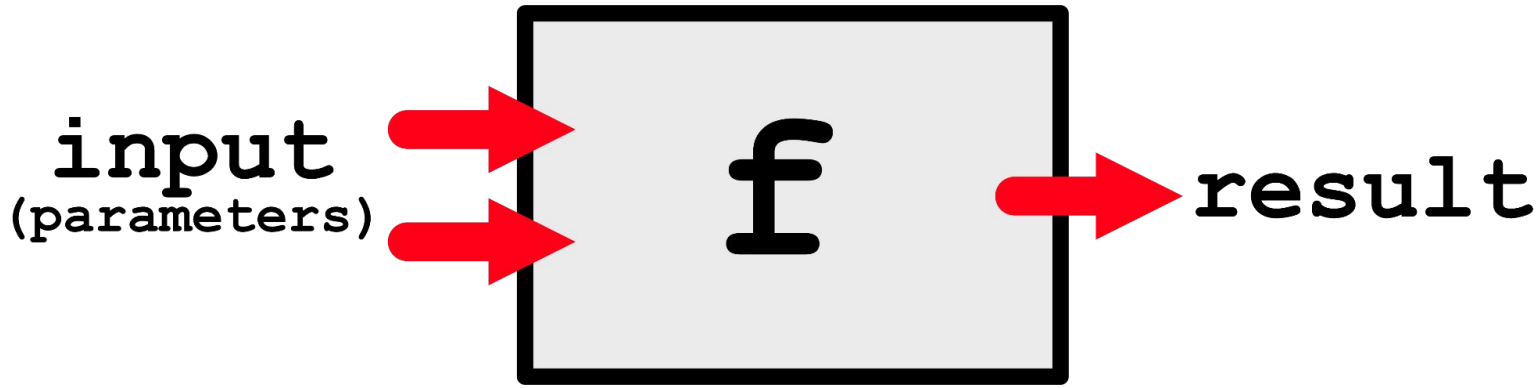
Helpful for ***debugging***

coding terminology

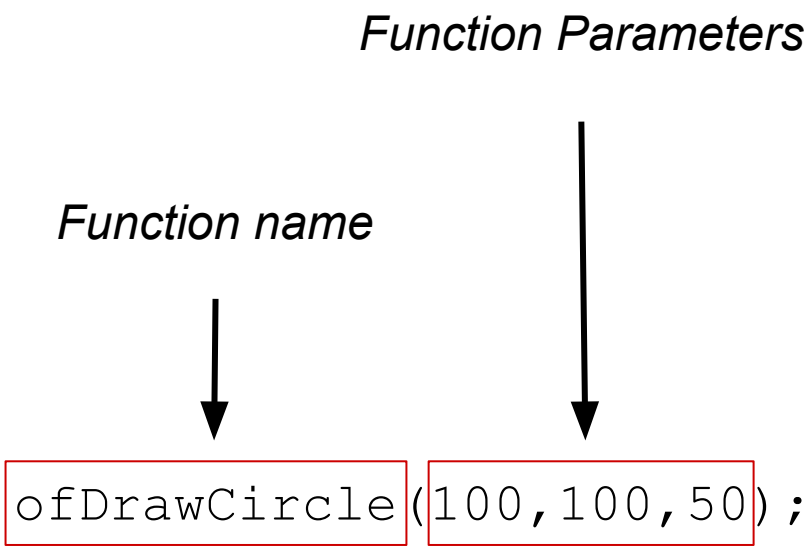
functions, variables & scope

Functions

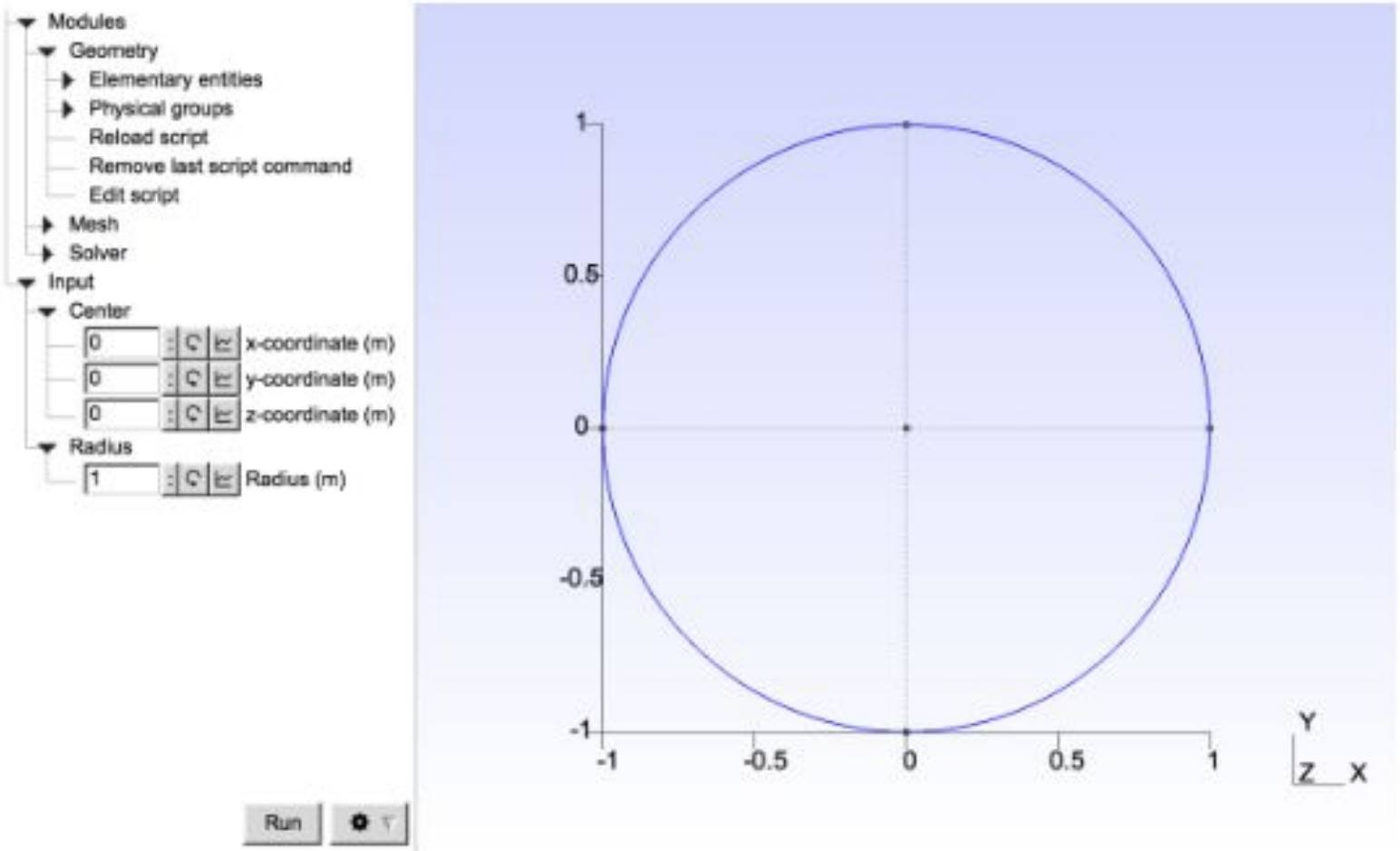
- A set of statements identified by a name, which takes input, does some specific computation, and produces an output result.
- You can think of a function as a box that takes in input parameters, performs some actions and gives back a result



A OF drawing function



- Sometimes functions DO something for us, like draw a circle but do not spit out a result


Modules
Geometry
Elementary entities
Physical groups
Reload script
Remove last script command
Edit script
Mesh
Solver
Input
Center
0
x-coordinate (m)
0
y-coordinate (m)
0
z-coordinate (m)
Radius
1
Radius (m)
Run
1
0.5
0
-0.5
-1
-1
-0.5
0
0.5
1
Y
Z
X

Documentation of an OF function

ofDrawCircle(...)

```
void | ofDrawCircle | (float x, float y, float radius)
```

Nothing, no result *(void)*

Function name

Function Parameters, 3 numbers *(float)*

- **Sometimes** functions DO something for us, like draw a circle but **do not** spit out a result
- This function requires 3 parameters
  - The “signature” of the function tells us these need to be numbers *(float)*

simple animation

Last week we did this:

```
ofDrawCircle(mouseX,mouseY,50);
```

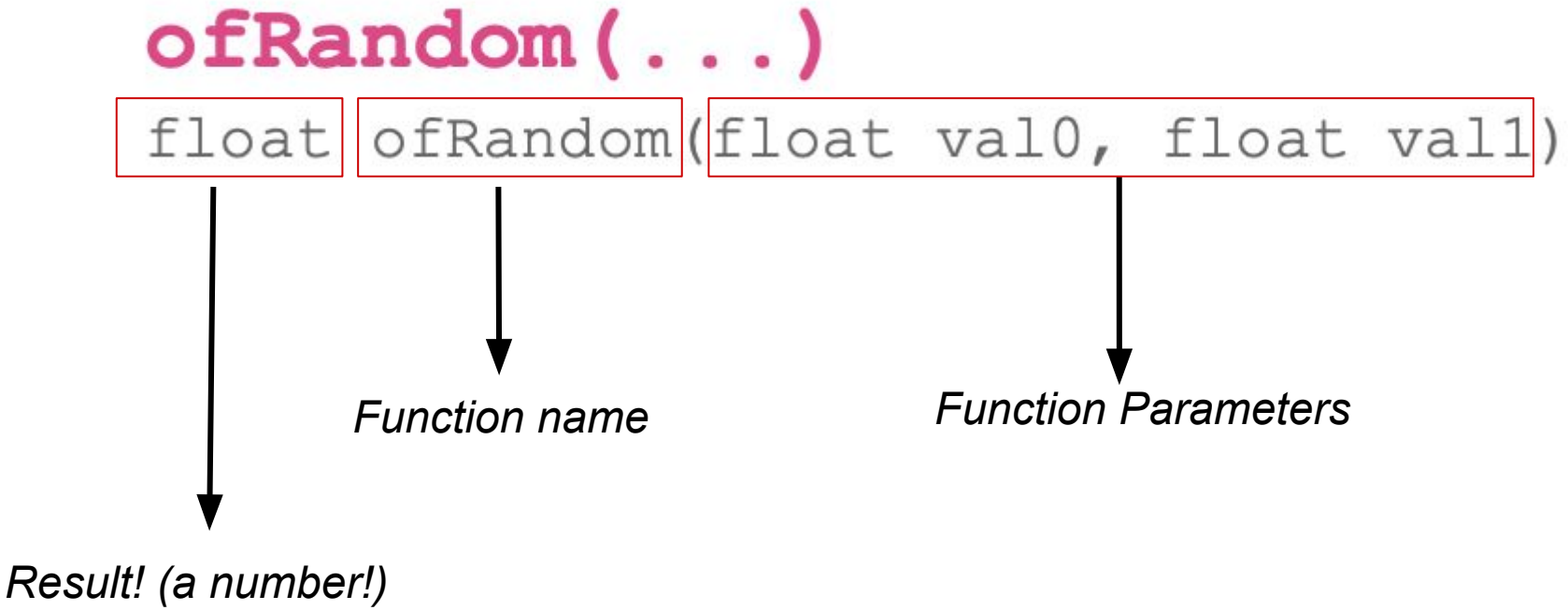
passing in the ***variable*** called `mouseX` as a parameter. `mouseX` is a number

This would spit an error!

```
ofDrawCircle("center of screen",
                          mouseY ,50);
```

NOT A NUMBER (a “string”)

Documentation of an OF function

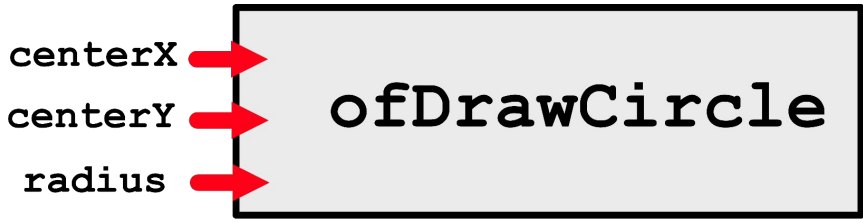


Sometimes functions take an input and DO spit out a result…

Let's try this:

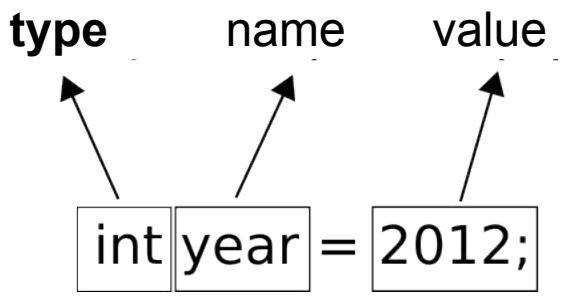
```
ofDrawCircle(mouseX,
                    mouseY + ofRandom(-100,100),50);
```

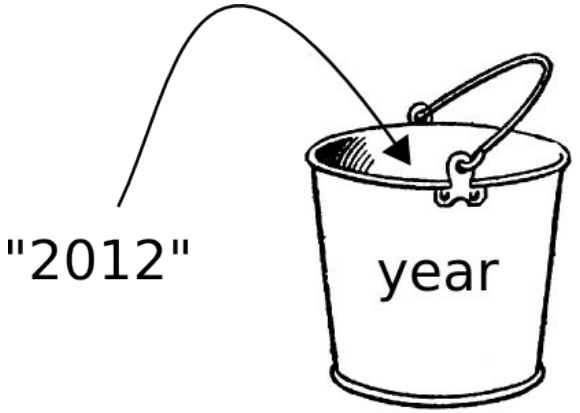
A number: the Y mouse position (variable)
+ a random offset (result of a function)



variables

- we run one command at a time so we need a place to store values until we need them again
- a layer of abstraction:
  - no need to know the location of a value in memory
  - I just need to know its name





- C++ is a **strongly typed language**
  - i.e. it is strict about declaring the **type** of variables

https://openframeworks.cc/ofBook/chapters/cplusplus_basics.html#variablespart1

- how to name variables:
  - Convention (in OF) is to use “camel case”, starting with lower case
    - E.g. myVar, centerOfRect, etc…
      - V.S. “snake case”: my_var, center_of_rect
  - **don’t** use spaces or special characters except “_”
  - **don’t** start names with numbers
  - start with minuscule
    - except classes (but we will cover these later)

Some variable types

Type	Description	Examples
`int`	An **integer**, a whole number without a fractional part	`0`, `-132`, `25`
`float`	A **floating point number**, a number with a fractional part	`0.0`, `-100.32`, `43.2f`
`char`	An (ASCII) character, usually described by single character bounded by **single** quotes	`'a'`, `'7'`, `'\n'`, `'$'`
`bool`	A Boolean value (either true or false)	`true` or `false`
`string`	Text, a C++ string of characters, bounded by **double** quotes. This is actually a "class"	`"Hello, world!"`
`void`	A special type used to indicate "nothing", e.g. used to indicate no return value for function	`void drawCircle()`

C++ docs: http://www.cplusplus.com/doc/tutorial/variables/

using variables

- *Defining (declaring)* the variable:

```
float x;
```

- *Assigning* a value to the variable:

```
x = 400;        // a number
x = y * 2;      // the result of an arithmetic operation
float z = 32;  // Declaring and assigning on the same line
```

- *Modifying* the value to a variable:

```
x = x + 400;
```

- *Reading and using* the value:

```
cout << x << endl;
ofSetLineWidth(x);
ofSetColor(x/2);
```

Some variable types

Numbers! But not all numbers are the same

Type	Description	Examples
`int`	An **integer**, a whole number without a fractional part	`0` , `-132` , `25`
`float`	A **floating point number**, a number with a fractional part	`0.0` , `-100.32` , `43.2f`
`char`	An (ASCII) character, usually described by single character bounded by **single** quotes	`'a'` , `'7'` , `'\n'` , `'$'`
`bool`	A Boolean value (either true or false)	`true` or `false`
`string`	Text, a C++ string of characters, bounded by **double** quotes. This is actually a "class"	`"Hello, world!"`
`void`	A special type used to indicate "nothing", e.g. used to indicate no return value for function	`void drawCircle()`

- `char` is actually a small number (8 bits)
- `'A'` gives back the ASCII code of the character A, in this case it is 65

Converting between numbers

Type	Description	Examples
`int`	An **integer**, a whole number without a fractional part	`0`, `-132`, `25`
`float`	A **floating point number**, a number with a fractional part	`0.0`, `-100.32`, `43.2f`
`char`	An (ASCII) character, usually described by single character bounded by **single** quotes	`'a'`, `'7'`, `'\n'`, `'$'`
`bool`	A Boolean value (either true or false)	`true` or `false`
`string`	Text, a C++ string of characters, bounded by **double** quotes. This is actually a "class"	`"Hello, world!"`
`void`	A special type used to indicate "nothing", e.g. used to indicate no return value for function	`void drawCircle()`

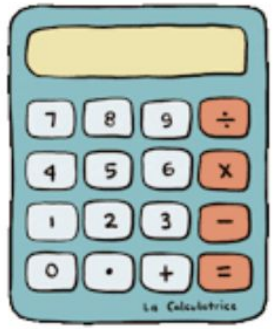
- It is fine to “convert” between different number types
- But each type has specific properties, e.g.:
  - Floating point number allow for decimal places so
    - `float x = 10;` results in x being 10.0 (10.0f to be precise)
  - Integers do not allow for decimal places so
    - `int y = 10.3213;` results in x being **10** (fractional part is gone)
  - Boolean values can be only `true` or `false`
    - See it like 0 or 1, so if `bool x = 10;` then `x` will be `true`

More (bigger) number types

Type	Description	Examples
`long`	A 64 bit **integer**	`0`, `-132`, `25`
`double`	A double precision (64 bit) **floating point number**, a number with a fractional part	`0.0`, `-100.32`, `43.2`

- Sometimes you will see a “qualifier” before a type
  - E.g. `unsigned int` means that the integer is strictly positive or 0
  - That is useful when dealing with memory at a lower level, no need to bother now
- `float` is 32 bit, which may not be sufficiently precise for certain applications (e.g. simulation)

Arithmetic operators



Operator	Result	Notes
`+`	Addition	
`-`	Subtraction	
`*`	Multiplication	
`/`	Division	
`%`	Modulo (remainder of division)	Applies only to integers

- Operators have **precedence**, with `*`, `/` and `%` being evaluated before `+` and `-`
- We can use **parentheses** to determine precedence, like `(a + b)*(c - b)`
- E.g. `a + b * c` **is not equal to** `(a + b) * c`
  - Useful to think of this by removing the `*`, so $a + bc$ vs $(a + b)c$

Some variable types

Type	Description	Examples
`int`	An **integer**, a whole number without a fractional part	`0`, `-132`, `25`
`float`	A **floating point number**, a number with a fractional part	`0.0`, `-100.32`, `43.2f`
`char`	An (ASCII) character, usually described by single character bounded by **single** quotes	`'a'`, `'7'`, `'\n'`, `'$'`
`bool`	A Boolean value (either true or false)	`true` or `false`
`string`	Text, a C++ string of characters, bounded by **double** quotes. This is actually a "class"	`"Hello, world!"`
`void`	A special type used to indicate "nothing", e.g. used to indicate no return value for function	`void drawCircle()`

Not numbers! (more later)

- `string hello=100;` Is **not** valid
- Likewise `float value="Hello";` or `void value="Hello"`

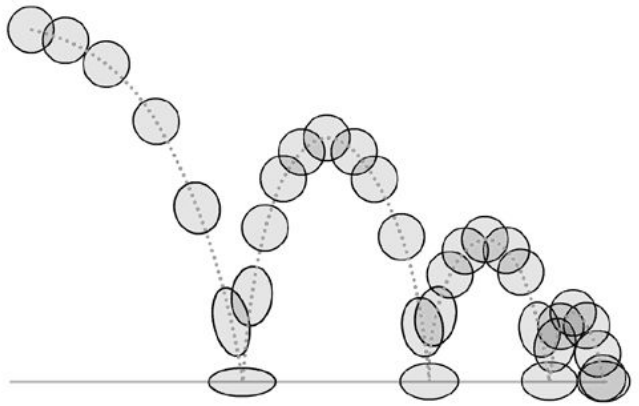
C++ custom types

- C++ allows us to define *custom types,*
  these are usually defined with “classes”
- The `string` type we saw earlier is an example of a class
  - It is available as what is known as the C++ “standard library”

C++ custom types

- Classes have attributes (properties) and methods (functionalities specific to the class). These are accessible with the “dot”, e.g:
  - `string myString = “Hello world”; // create a new string`
    `int length = myString.size();  // the string length`
- We won’t look at how to define classes until later in the course
- But we have a custom definition right under our eyes
  - In every class example
  - `ofApp` is a class and it is defined in ofApp.h
  - It’s methods are implemented in ofApp.cpp
- OpenFrameworks define many other class types, e.g `ofColor`

Class Ball



- Attributes
  - shape
  - color
  - weight
  - material
  - content

- Methods (behaviors)
  - movement
  - bounce
  - inflating
  - deflating
  - blowing up

Lifetime of variables: What is “scope”?

{ It's all about braces }

- In C++ braces are used to delimit blocks of code
  - E.g. the contents of a function
  - The statements that follow certain *keywords*
    - E.g. `if`, `else` (we will look at these today)
- Braces define the *scope* of a variable

{ It's all about braces }

- If a variable is defined between a pair of **{ . . . }**, E.g. within `ofApp::draw(){....}` it will only be accessible within that pair
  - Not in a separate pair, e.g `ofApp::update(){....}`
  - But it will be accessible in nested pairs **{ { } { { } } }**:
    - ```
      ofApp::draw() {
         int value = 10;
          if (ofGetMousePressed()) {
           cout << value << endl;
          }
      }
      ```
      ```

{ It's all about braces }

- If a variable is defined outside of braces in a .cpp file, it will be available to all {}'s within that file
  - It is a "global" variable
  - We won't cover now how To access it in different files, but it is possible

```
1  #include "ofApp.h"
2
3  double t = 0.0;
4
5  //--------------------------
6  void ofApp::setup(){
7
8  }
9
10 //--------------------------
11 void ofApp::update(){
12   std::cout << t << endl;
13 }
```

{ It's all about braces }

- To keep it simple, you can use this to define variables that persist during you program execution
  - Warning: it may make a pro C++ coder shrug
  - But simple and fine with me:)

```
1  #include "ofApp.h"
2
3  double t = 0.0;
4
5  //--------------------------------
6  void ofApp::setup(){
7
8  }
9
10 //--------------------------------
11 void ofApp::update(){
12   std::cout << t << endl;
13 }
```

{ It's all about braces }

- A less "shruggy" way to define variables available to your app:
  - Define in ofApp.h
    inside the {...} of the "class definition"
    - I.e.

```
class ofApp : public ofBaseApp{
  // define variables here, e.g.
  double t;

  public:
    void setup() override;
    void update() override;
    void draw() override;
    void exit() override;
```

  - Set their values in `setup` ofApp.cpp
    - I.e.

```
void ofApp::setup(){
  t = 0.0;
}
```

  - These will be available across all methods of ofApp

Global Scope

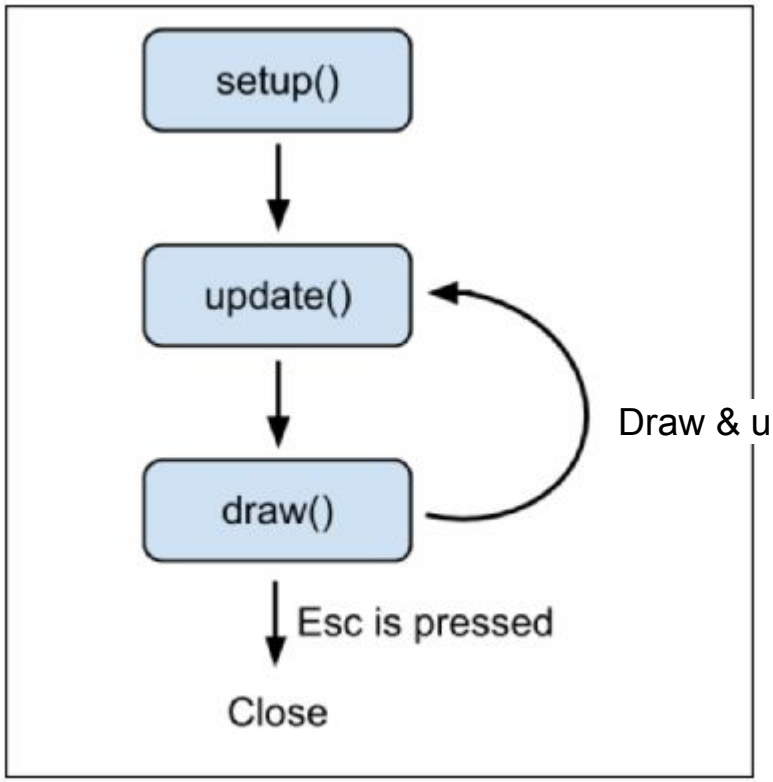
🌎 Global Scope

E.g. globally available functionality

Scope of our ofApp

- Scope of Setup
- Scope of Update
- Scope of Draw

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



BREAK!
(10 mins)

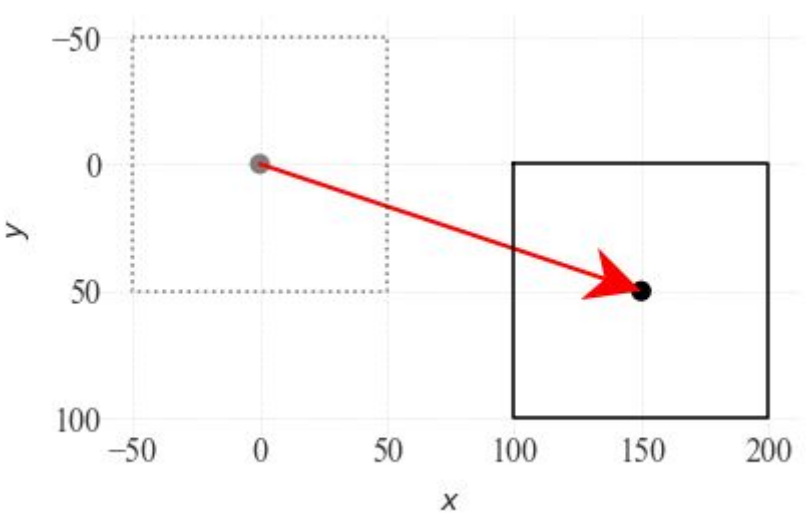
Putting some variables to use

2D transformations

Commands (for now)

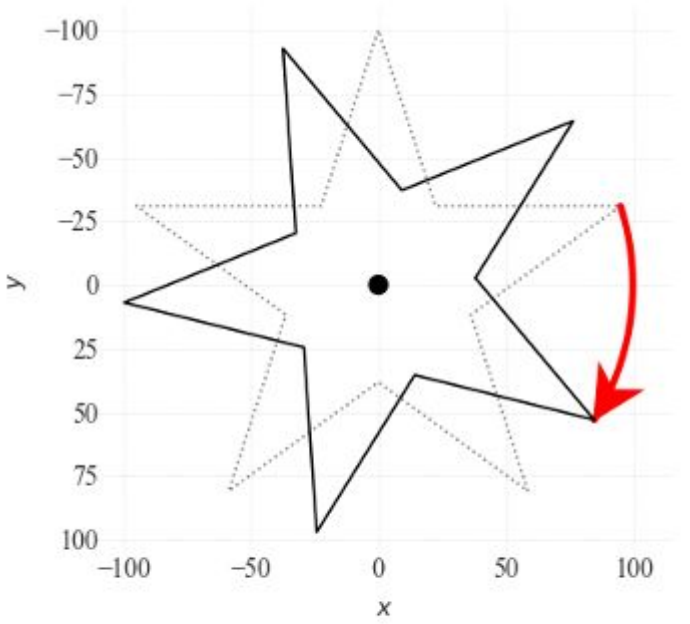
- `ofTranslate(x, y);` move in x, y direction
- `ofRotateDeg(degrees);` rotate degrees
- `ofRotateRad(radians);` rotate radians
- `ofScale(amt);` uniform scale
- `ofScale(xAmt, yAmt);` scale in x and y direction

```
ofTranslate(150, 50);
ofDrawRectangle(-50, -50, 100, 100);
```

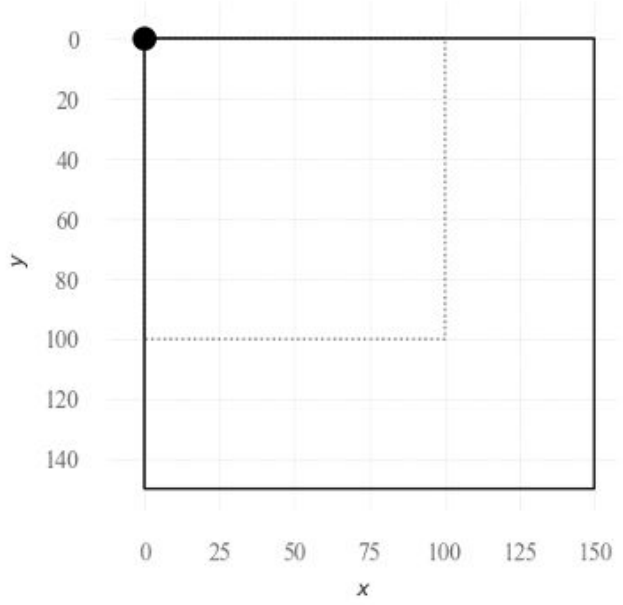


Same as:

```
ofDrawRectangle(-50 + 150, -50 + 50, 100, 100);
```




```
ofRotateDeg(50);
drawStar(0,0);
```




```
ofScale(1.5);
```

More in week 4!

DEMO

void return means it's just *doing* something

Sometimes functions DO something for us, like translating (moving, dislacing) the subsequent drawing commands

ofTranslate(...)

```
void ofTranslate(float x, float y, float z)
```

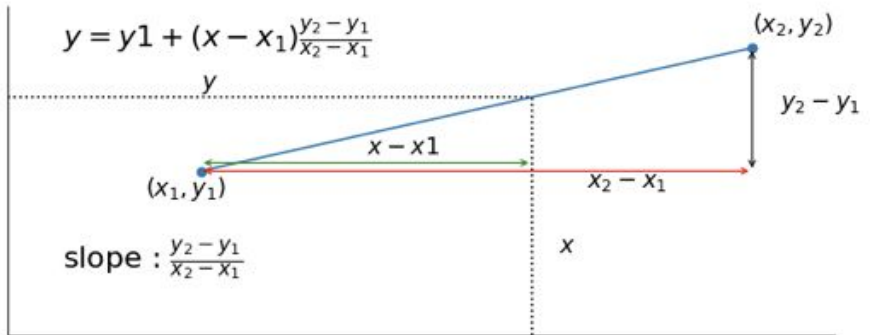
`z` is optional!

https://openframeworks.cc/documentation/graphics/ofGraphics/#show_ofTranslate


The void!
void returns nothing

More use of variables. Linear interpolation and `ofMap`

- **Say we want to change the size of a rectangle following the mouse, so it shrinks as the mouse is farther from the center of the ofApp window**
  - We can compute the distance of the mouse x and y positions to the center with
    ```
    float dist = ofDist(mouseX, mouseY, ofGetWidth(), ofGetHeight());
    ```
  - `dist` is always greater or equal to 0
- **And let's define an input range:**
  - minimum and maximum distance `minDist` and `maxDist`.
- **Let's define an output range:**
  - a minimum size and a maximum size `minSize` and `maxSize`.
- With this information,
  mapping distance to size can be done with a linear interpolation:
  ```
  float size = minSize + (dist  - minDist)*
                  ((maxSize - minSize)/(maxDist - minDist))
  ```



More use of variables. Linear interpolation and `ofMap`

- **Can be hard to remember.**
- **The `ofMap` function comes to the rescue:**
  - **It maps an input range to an output range**

What *types* are expected as parameters?

`ofMap(...)`

```
float ofMap(float value, float inputMin, float inputMax, float outputMin, float
outputMax, bool clamp=false)
```

It returns a float to us.

What does ofMap return?

What *types* are expected as parameters?

ofMap(...)

```
float ofMap(float value, float inputMin, float inputMax, float outputMin, float
outputMax, bool clamp=false)
```

It returns a float to us.

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

demo

Some conditionals

Our friends the
curly brackets

```
if ( ????? )
{
   //then run this block of code
}
```

`if` within our ofApp.cpp

- we run some commands some of the time and not others
- different behaviours depending on conditions
- Think of `if` as a function that accepts `one bool`

```
void ofApp::draw()
{

    if(bool)
    {


    }

}
```

`if` within our ofApp.cpp

- Draw only when mouse down

```
void ofApp::draw()
{

        if(ofGetMousePressed())
        {


        }

}
```

ofGetMousePressed(...)

`bool ofGetMousePressed(int button)`

https://openframeworks.cc/documentation/events/ofEvents/#show_ofGetMousePressed

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`

“==” different from “=”

```
if (x == y) {   // do x and y have the same value?
}
x = y;          // assign value of y to x

if (x = y) { // COMMON ERROR,
// ASSIGNS Y TO X!!!
}
```

Conditional statements - if

```
if ( raining ) // evaluates statement
{
   // if it's raining it runs the following
   code
   - take umbrella
   - take car
}
```

how would we draw a rectangle when mouseX is past the mid-point of the x-axis and a circle when it's before?

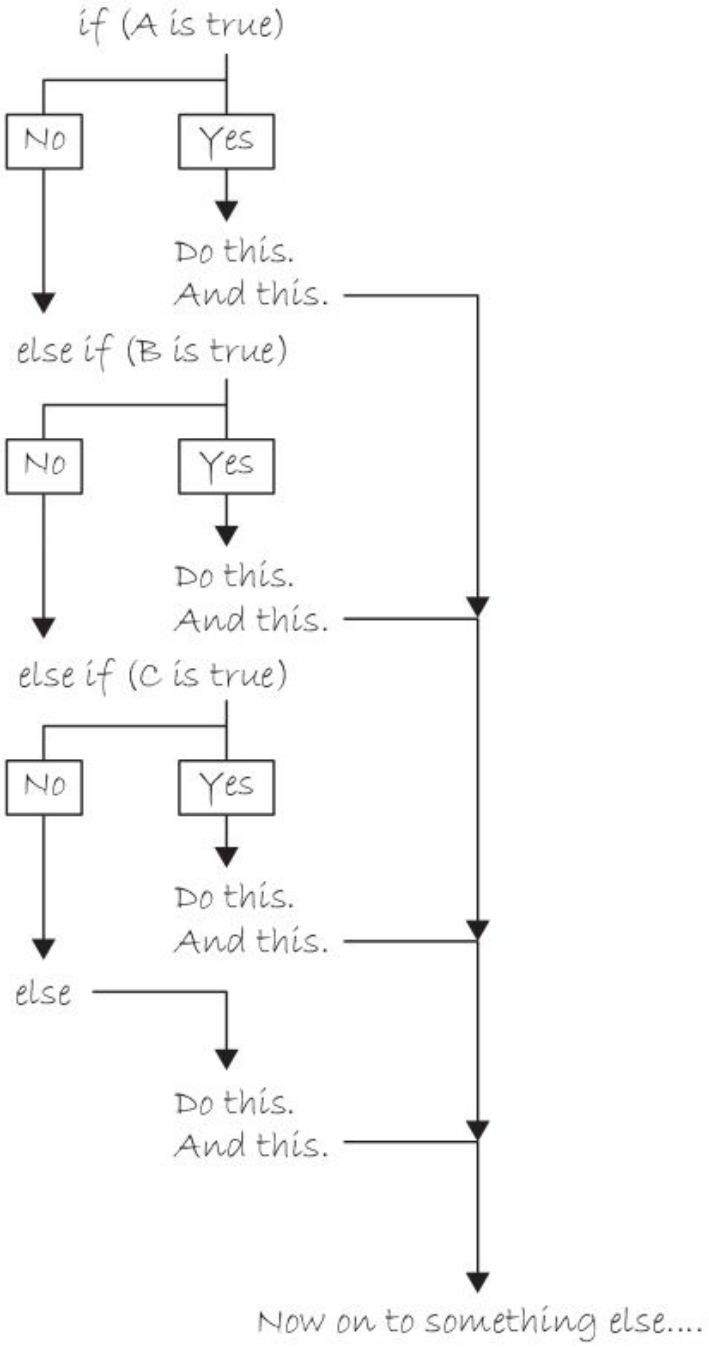
Conditional statements - `if / else if`

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

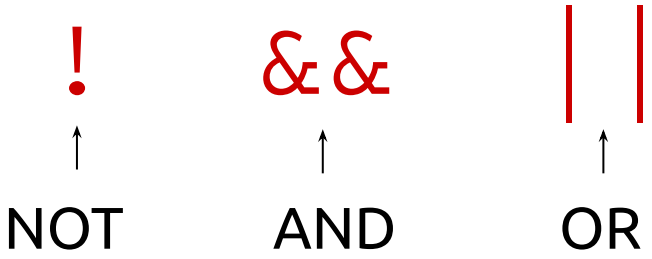
Conditional statements - `if / else`

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

- the
- `if/else if`
- path



Logical operators:

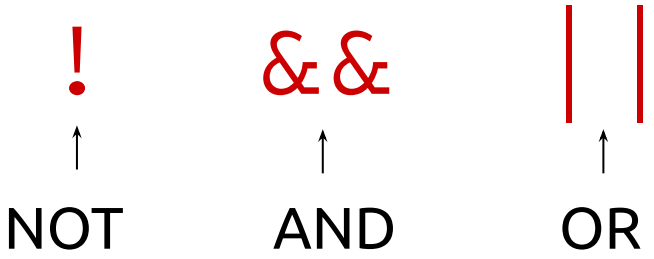



```
if ( !raining )  // IF it's NOT raining
{
     - take bicycle
}
```

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

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

Logical operators:



- Logical operators are used to compose expressions made of boolean (`true` or `false`) values
- Similar to English:
  - AND/OR (`&&` and `||`) are always applied to two values, one on the left and one on the right, e.g:
    - `a || b` or `a && b`
  - NOT is applied to the left and "flips" the meaning, e.g.:
    - `!true == false`
- See https://github.com/colormotor/PFAD/blob/main/docs/reference.md For more info on this and as a reference for the class

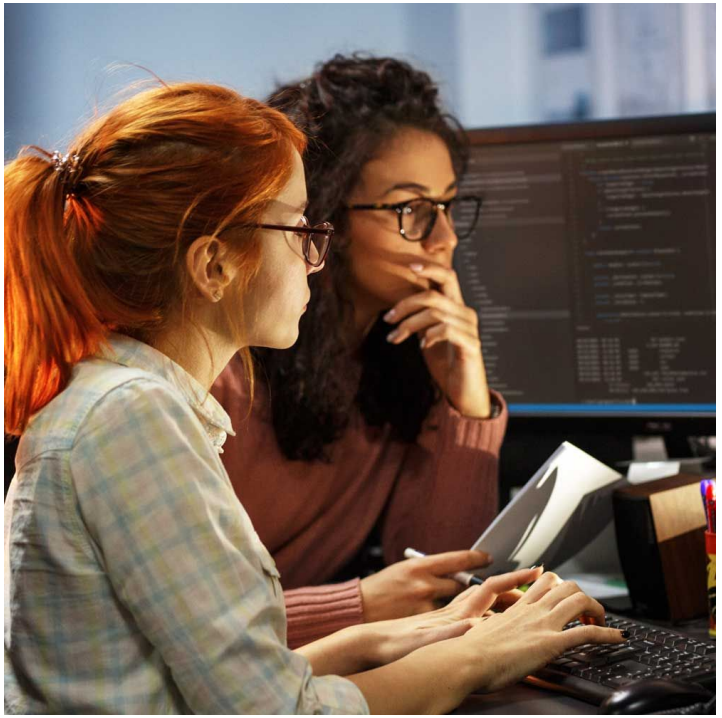
Homework

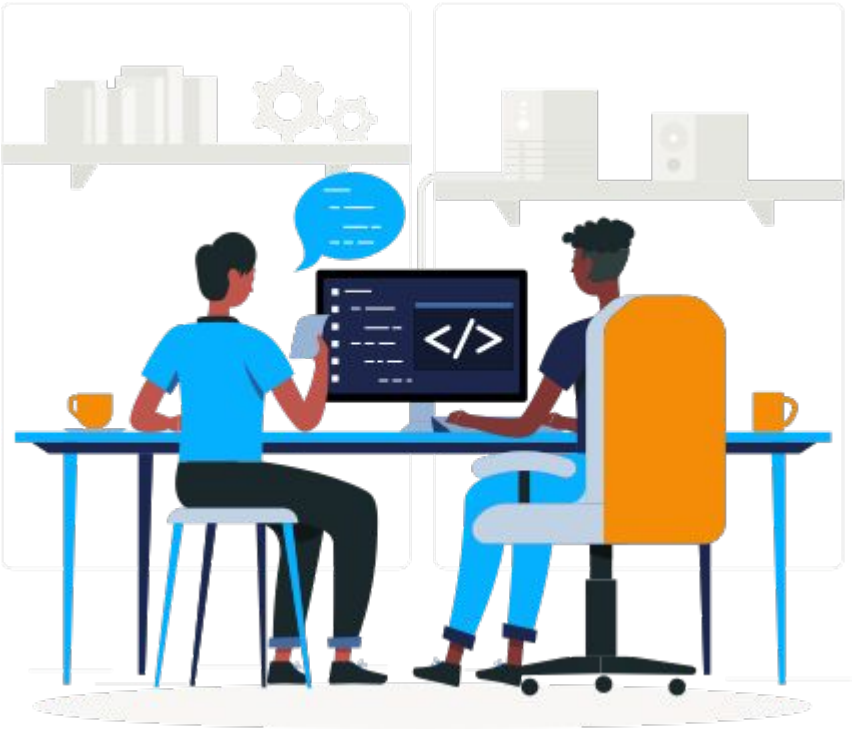
Code a drawing program

- think about color
- use transformations
- Use transparency
- Create complexity

(lab code are *tools* to use)

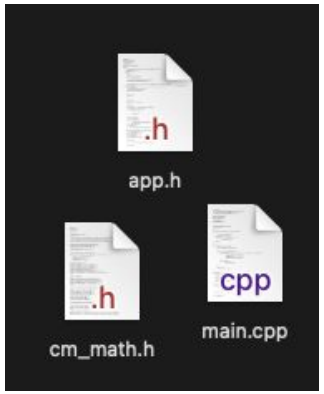
Pair programming

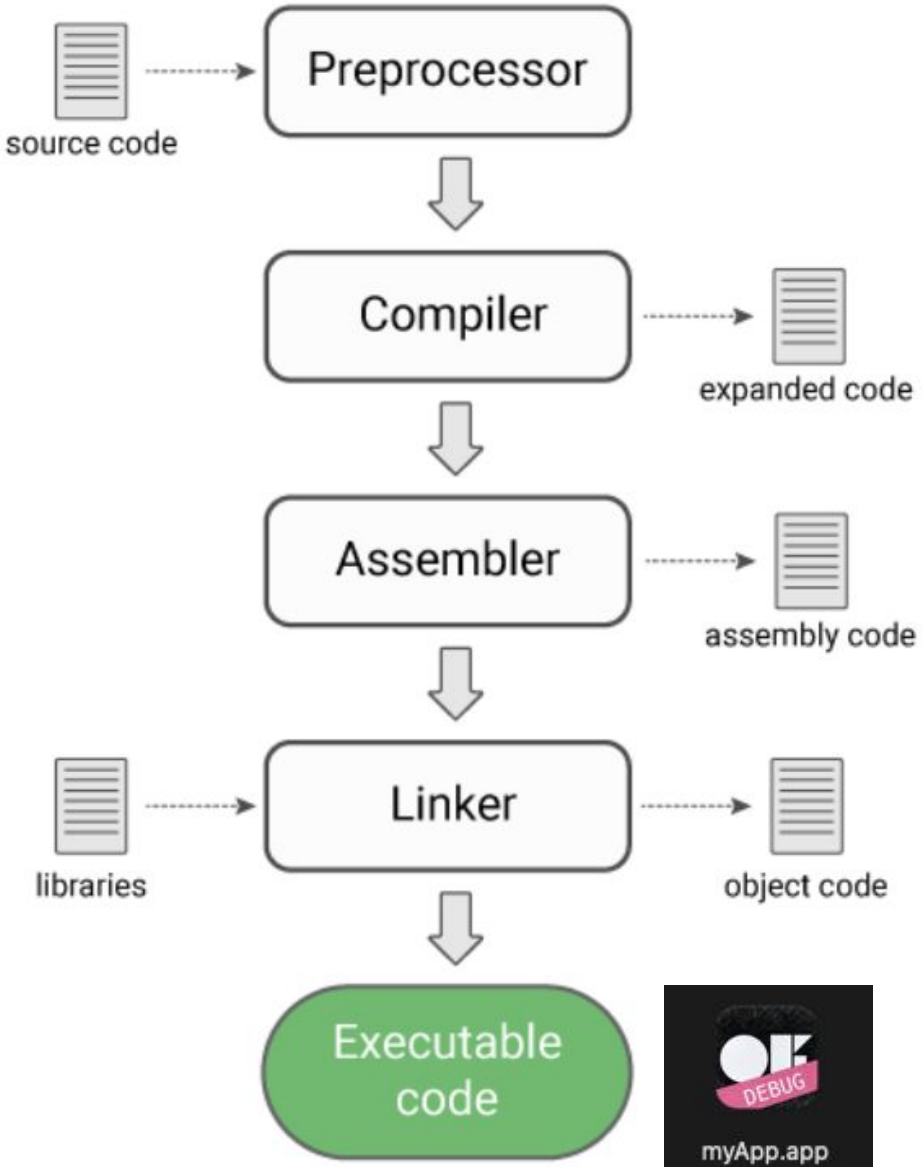




- **Driver:** types code // reads instructions
- **Navigator:** reads instructions // looks up commands // corrects code

How does C++ work?





A quick look at the preprocessor (wherever you see #)

- As the name implies, it **preprocesses** files before compilation occurs
  - Essentially it uses directives starting with # to make modifications to the code before compilation
  - Remember: when you see # in C++, the preprocessor is involved
- `#include` includes the contents of the specified file into the file where it is specified
  - `#pragma once` makes sure this happens only once

ofApp.h

```
#pragma once

#include "ofMain.h"

class ofApp : public ofBaseApp{

  public:
    void setup() override;
    void update() override;
    void draw() override;
    void exit() override;

    void keyPressed(int key) override;
    void keyReleased(int key) override;
    void mouseMoved(int x, int y ) override;
    void mouseDragged(int x, int y, int button) override;
    void mousePressed(int x, int y, int button) override;
    void mouseReleased(int x, int y, int button) override;
    void mouseScrolled(int x, int y, float scrollX, float scrollY) override;
    void mouseEntered(int x, int y) override;
    void mouseExited(int x, int y) override;
    void windowResized(int w, int h) override;
    void dragEvent(ofDragInfo dragInfo) override;
    void gotMessage(ofMessage msg) override;

};
```

ofApp.cpp

```
#include "ofApp.h"

//--------------------------------------------------------------
void ofApp::setup(){

}

//--------------------------------------------------------------
void ofApp::update(){

}

//--------------------------------------------------------------
void ofApp::draw(){

}

//--------------------------------------------------------------
void ofApp::exit(){

}

```

ofApp.cpp after `#include "ofApp.h"`

```
#include "ofMain.h"

class ofApp : public ofBaseApp{

  public:
    void setup() override;
    void update() override;
    void draw() override;
    void exit() override;

    void keyPressed(int key) override;
    void keyReleased(int key) override;
    void mouseMoved(int x, int y ) override;
    void mouseDragged(int x, int y, int button) override;
    void mousePressed(int x, int y, int button) override;
    void mouseReleased(int x, int y, int button) override;
    void mouseScrolled(int x, int y, float scrollX, float scrollY) override;
    void mouseEntered(int x, int y) override;
    void mouseExited(int x, int y) override;
    void windowResized(int w, int h) override;
    void dragEvent(ofDragInfo dragInfo) override;
    void gotMessage(ofMessage msg) override;

};

//--------------------------------------------------------------
void ofApp::setup(){

}

//--------------------------------------------------------------
void ofApp::update(){

}
```

Again why bother with the preprocessor

- The template is built so you can easily switch to normal animation
  - I.e. clearing the screen
- Due to the OF bug this requires a special setup:
  - In ofApp.h, we have `#define AUTO_CLEAR 0`
    - This will tell the preprocessor to replace any occurrence of "`AUTO_CLEAR`" to the value the follows
  - In main.cpp this is handled but since we have different code for the cases of `AUTO_CLEAR` we use `#define` again to specify the desired window width and height
- In short, to change the window width and height, in main.cpp change the values in these lines:
  - `#define APP_WIDTH 640`
  - `#define APP_HEIGHT 480`