

# THE ARTIST AS PROGRAMMER

AHD-2241-A / VCD-2241-A

Thursday, 12:00pm-1:00pm

Fall 2020

Justin Elm

jelm1@sva.edu

Syllabus, introduction, basic techniques

"The problem with programming is not that the computer isn't logical—the computer is terribly logical, relentlessly literal-minded. Computers are supposed to be like brains, but in fact they are idiots, because they take everything you say at face value." -Ellen Ullman, *Life in Code*

What are we going to cover?

HTML, CSS, JavaScript, jQuery, and some other  
interesting things

...and then we'll see where we are in the semester

If you haven't already, take the time to read through the syllabus on Canvas.

Something to keep in mind this semester...

This semester will sort of be like high school math...

The analogy I'm trying to make is that every class builds on the last class.

Why is that important? If you miss a class, or skip a project, or you don't do an assignment...

You will fall behind! This is not something you can check in and out of, it will be difficult.

So if you find you're falling behind, or getting confused, or overwhelmed, reach out so we can solve the problem.

Don't suffer in silence!

This is particularly important due to the semester being online. I'm not going to be able to read the room, or see how you're doing in person.

This makes open communication even more important.



Additionally, ask questions! Please ask as many questions as you possibly can, it will help everyone. There's a "General Questions" discussion thread for each week within the Canvas course shell—please make use of this frequently.

Before we start jumping in, I want to mention a few common things many individuals suggest they wish they knew before starting to learn to code.

*1. It's much easier to learn to code if you have an end goal in site.*

What do you want to build? How should it look?

The point is, with a clear end goal in mind, the project will naturally create questions and problems to solve.

And those solvable problems are where you'll do the most learning. So start thinking now.

So, today you should be thinking about "what do I want to get out of this class?"

You should be thinking about this every semester, every year, and every class you're enrolled in, because it will help guide your learning.

*2. Programming is a skill, just like any other skill.*

Just like painting, drawing, animating, etc., there are techniques and tools and best practices that have been developed over time. You can use these, modify them, break them, etc.

The point is I'll do my best to communicate as many of these ideas and tools and techniques to you as I can, but to really learn you need to do it yourself.

No matter how long you watch someone draw, you're not going to get any good unless you sit down and draw.

And! I want to dispell the myth that this is math heavy, or intimidating, or only for a specific type of person can do this—all of that is incorrect.

Anyone can learn to draw; anyone can learn to code. There's no point in being intimidated.

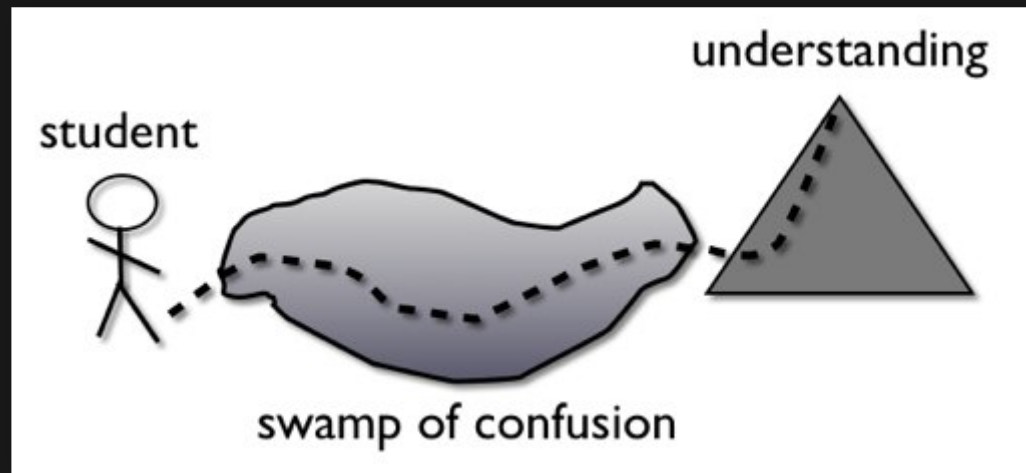
*3. It's not going to work on the first try (or maybe even the second or third).*

Don't be discouraged or get annoyed, that's normal and to be expected.

To be honest, this is an experience that everyone has!  
Including veteran programmers.







*4. Sticking with it is the most important thing.*

Sometimes progress comes quick and slows down,  
sometimes it's the opposite.

Eventually everyone hits a wall, and it gets really hard,  
but the only way to learn is to keep going.

# TOOLS AND WORKFLOW

Before talking about anything else, let's talk about a few basic things.

While we are starting from scratch from a programming perspective, we're not starting from scratch from a computing perspective.

What does mean? There's an expectation that you have familiarity with computers in general.

I'll be working in a macOS environment, but you may be using something else like Microsoft Windows or a Linux distro.

Let's also take a second to just explain, in the most basic terms, how a website works.

We're all hopefully familiar with typing in a website address and getting results, but what's actually going on?

So, what is a website?

A website is just a collection of files (code files, image files, font files, etc.) connected together in a folder...  
that's it.

(That's not *really* it, but stick with me here)

How those files interact, connect, and how they  
behave is up to us

## What is a web browser?

The web browser is how we view what's on the internet. The browser takes our requests for what we want to see and then renders the code into a more human readable format.



## What is a server?

A server is a computer connected to the internet, so that other computers can ask it for data. Every server connected to the internet has a unique IP address, which functions in a similar way to a physical address in the real world.

## What is a domain name?

It's a human readable name that corresponds directly to an IP address, so that when we type <https://google.com/> in the browser, the browser changes that request into an IP address, which contacts the computer located at that address, and the computer returns the requested information (the website files)

So, for the basics, to create a website you need:

1. Website core files
2. A host to provide server space (for your files)
3. A domain name (if you want the website to be publically accessible)

To view a website

1. An internet connection
2. A web browser

As a basic principle, all we're doing is moving information and rendering it.

In programming there are an infinite number of tools and workflows one can adopt to be a productive programmer. There are positives and negatives of each, but for our class it's best for all of us to use the same tools.

So, first things first, choosing a web browser. The browser we use is particularly important because it both renders our projects and offers tools to develop our projects.

For this class we're going to be using Google Chrome. The primary reason are the developer tools (which we'll cover later) and the way in which they're integrated into the browser. Granted, Chrome is a bit memory hungry but it also has a great JavaScript engine.

To see what I mean, go ahead open Chrome

At the top, in the menu bar, navigate to:

View > Developer > Developer Tools



Here we find all sorts of useful pieces of information.  
Take a second to click around and see what you can  
find.

We have a number of tabs, including Elements, Console, Sources, Network, Performance, and More

We're primarily concerned with Elements and Console

Elements will show us all the HTML elements on the page.

Console relates to the JavaScript console, and we'll use this to see what our code is doing.

Over the semester we'll delve deeper into this, but we're just giving a brief overview.

Where do we actually write the code? And what's the difference between a text editor and an interactive development environment?

A text editor is pretty straight forward, its function is in the name, it just edits text.

What's that mean? It means that technically you can program using anything from Microsoft Word to TextEdit to a purpose built text editor.

Some common text editors are Atom, Sublime Text, and TextMate

An interactive development environment (IDE) has significantly more tools, like an integrated console, linter, debugger, autocomplete and function lists, etc.

If you end up programming beyond this class, you'll want to adopt an IDE because it *will* make your life easier, but for us we want to go for the basics and eliminate extra information so as to not be overwhelmed.

In this class we'll be using [Atom](#)--it's simple, reliable, and aesthetically pleasing.

When we get stuck into JavaScript we'll also make use of [Repl.it](#) because it will offer a lot of features, of which we'll see later.

Side note, [REPL](#) is actually a programming concept, hence the name.

# DEBUGGING!

Probably the most important thing to remember today



## What is debugging?

Debugging essentially equates to trying to figure out why something doesn't work, or what is going on, in our code?

There could be syntax errors, or logical errors, etc.

Sometimes there's an error message, often times there isn't, so how do we find them?

First things first, how to get at those error messages?

Chrome:

View > Developer > JavaScript Console || Command +  
Option + J

Safari:

Safari > Preferences > Advanced > Show Develop  
menu...

Develop > Show JavaScript Console || Command +  
Option + C

Firefox:

Tools > Web Developer > Web Console || Command +  
Option + K

Note, those are all for the JavaScript console.

You can also see the HTML source/elements, which we'll also use in the future. You can get to those tabs in a browser in a similar way, hotkeys are different.

With this, checkout the files `index.html` and `debugger.zip` in the Week 1 module on Canvas.

Download those files and open them in one of those editors mentioned before.

Focus on the file titled index.html, what do we see?





Drag the HTML file into a web browser, what's rendered?

Open the developer console, can you see the message?

Focus on the debugger.zip file, decompress it, and see the debugger.html and debugger.js file.

Take a second to look at line 18; this is how we link our HTML and our JS.

Note, `debugger.html` and `debugger.js` must be within the same folder, adjacent to each other, so the files can recognize each other.

In the html file:

```
<h3>Does this work?</h3>  
<p id="demo">Replace these words.</p>
```

In the js file:

```
function replaceTheWords() {  
    document.getElementById("demo").innerHTML = replacementWords;  
}  
replaceTheWords();
```

Open the html file in a web browser.

Click on the text "Does this work?"—did the text change?

Read through the code, see if you can make any sense of it, Google what doesn't make sense.

# DATA TYPES!

What is that?



A data type is a way of classifying or identifying different types of data

With data types, we can determine:

- possible values for that type
- operations that can be performed on that type
- the meaning of the data
- the way values of that type can be stored

Most programming languages have the same data types (there's always a caveat)

- strings: word or sentence surrounded by "quotes"  
-- 'kittens are soft'
- integers: any number without a decimal point -- 7
- floats: any number with a decimal point -- 1.03
- booleans: true or false
- arrays: collections of data -- []
- associative array: collection of data with key-value pairs -- ['a': 200, 'b': 300]
- objects: a representation of something

## Array:

```
[ 'dogs', 'cats', 'turtles', 'birds' ]
```

## Object:

```
var fakeObject = {  
  color: "blue",  
  fabric: "cotton",  
  size: "M"  
};
```

In JavaScript, there are five primitive data types

- string
- number
- boolean
- undefined
- null

# Everything else is an object

- arrays, [ ]
- functions,
- objects, {},

```
function nameOfFunctionWeInvented() {  
  // do cool things  
}
```

```
{  
  name : 'Justin Elm'  
}
```

There's also a core set of language elements:

- operators: +, -, ===
- control structures
- statements

```
if ( somethingIsTrue ) {  
    // do something else  
}
```

```
var x = 1;
```

**ALRIGHT, VARIABLES!**



Variables are used to store data in the computer's memory so we can use it later

Always use the var keyword to declare a variable

```
var a;  
=> a;
```

Should return undefined

We need to take a second to talk about ES5 and ES6

ES stands for ECMAScript, and ECMA as defined:

Since 1961 and continuing in full force today, Ecma International® facilitates the timely creation of a wide range of global Information and Communications Technology (ICT) and Consumer Electronics (CE) standards

In essence, ES5/ES6/ES7 etc. are versions of JavaScript that have different standards.

Don't worry, you don't have to worry about this too much yet.

Except right now, for variables...

In ES5, we declare variables using the **var** keyword:

```
var a;  
=> a;
```

In ES6, the keywords **let** and **const** are introduced:

```
let a;  
=> a;  
const b;  
=> b;
```

The reason variable declarations changed, has to deal with scope, but that's a concept we'll cover later.

But a brief intro, is that **var** is *function scope* while **let** and **const** are *block scope*.

For example, if we declare:

```
var name = "Justin";
```

And then a few lines later in our code we write:

```
var name = "Martha";
```

What happened? We overwrote our variable... which we almost never want to do. We lose that original piece of information.

If we do the same thing:

```
let name = "Justin";
```

And then a few lines in our code we write:

```
let name = "Martha";
```

We'll get a warning in the console that says that variable has already been declared!

Note, you can *update* a **let** variable:

```
// outside of loop below, so scoped to window
let someInstance = "true";

if( someInstance === true ) {
    // inside the loop, so scoped to
    // block (curly brackets) so this is allowed
    let someInstance = false;
}
```



For **const**, we never update that variable. It's a *constant*, and immutable.

Example:

```
const name = "Jeeves";  
name = "Bartholomew";
```

We'll get an error, it won't let us do that.

By convention, all names should be camelCased,  
whether they're variables or functions

There are also certain words we can't use for naming,  
like var, function, class

```
var myScore = 100;  
var myString = "Hello World!"
```

Values are assigned to a variable using the = operator


```
var x = 1;  
x;  
=> 1
```


We will go over these concepts over and over again, so if you're feeling a bit overwhelmed or confused, that's okay.



LEARNING IS HARD.  
I DON'T TRUST ANY ONE  
WHO SAYS THINGS LIKE

XXX MAKES CODING EASY  
AND FUN. MASTER IN 7 DAYS

IT DOES MORE HARM   
THAN HELP BECAUSE  
IT BUILDS FALSE EXPECTATIONS  
AND DISCOURAGES SMALL

FAILURES, WHICH IS A GOOD  
THING, IMHO. 

NOW THAT I THINK ABOUT IT,  
I DID NOT LIKE LEARNING  
TO PROGRAM, MORE THAN  
PROGRAMMING ITSELF.

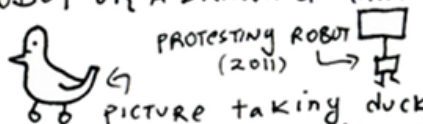


NOT BEING GOOD AT SOME-  
THING, AND TO BE LOST, <sup>AND TO</sup>  
<sup>KEEP</sup>  
<sup>doing it</sup>  
TAKES A LOT OF COURAGE.

I DON'T LIKE PROGRAMMING.  
IT'S DRY, SLOW, AND TEDIOUS.



BUT I LIKE MAKING THINGS  
THAT DO THINGS, LIKE A SIMPLE  
ROBOT OR A DRAWING THAT MOVES.



AND A LITTLE BIT OF CODE HELPS  
IT BECOME MORE INTERESTING

"I DON'T KNOW WHAT TO DO."

"I DON'T KNOW WHY THIS WORKS."

IF YOU ARE TELLING YOURSELF  
SOMETHING LIKE THAT, JUST KNOW  
I FEEL THAT ALL THE TIME.



MAYBE I JUST GOT A BIT USED  
TO BEING FRUSTRATED.  
I HAVE NO IDEA WHAT I'M DOING.  
AND I LIKE IT.



tcho8

Feel free to hit me up any time with questions.