

THE ARTIST AS PROGRAMMER

AHD-2241-A / VCD-2241-A or AHD-2241-B / VCD-2241-B

Wednesday or Thursday, 12:10pm-3:00pm

Spring 2019

Justin Elm

jelm@sva.edu

212.592.2255

Syllabus, introduction, basic techniques, data types,
variables, debugging

"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

Reminder to go over syllabus right now...

Something to keep in mind this semester...

This semester is like high school math (WE'RE NOT DOING MATH)

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

Why is that important? If you're late, if you miss a class, 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.

This year we're going to try something new, primarily
influenced by time

In the past we've spent a lot of time on HTML and CSS,
which is all well and good, until we run out of time for
JavaScript and jQuery

Technically speaking, HTML and CSS are not
programming languages, they're markup languages

HTML and CSS will come, JavaScript is much more
difficult

That being the case...we're going to leverage some free alternatives:

Learn HTML

Learn CSS

Learn HTML Due : January 23

Learn CSS Due : January 30

The point of doing these tasks online is to bring us all to the same page without spending valuable class time on the absolute basics.

Before we start talking about data types, 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, but more so in this course.

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 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.

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;
```

Okay, let's try this in repl.it or node (if it's on these machines)

you may have to make an account, or use gmail or something

**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.

Most websites are only starting to support and move from ES5 to ES6.

For our purposes in this class, we'll almost always use **var**.

I just want you to know that if you're searching around for help, and you see **let** and **const**, that's why.

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
```


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, let's do a little demo together.

Reminder: use `debugger.js` and `debugger.html` (for me)

Let's look at where we're going, then we'll do it together.

In an html file:

```
<h1>Does this work?</h1>  
<p id="demo">Replace these words.</p>
```

In a js file:

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


Alright, let's do the demo.


Hopefully we got to the end and didn't run out of time.



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.

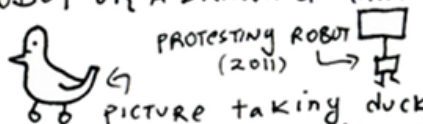


NOT BEING GOOD AT SOME-
THING, AND TO BE LOST, ^{AND TO}
^{KEEP}
^{doing it}
TAKES A LOT OF COURAGE.

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.

You're welcome to stop by my office at any time as well.

This building, 6th floor, room 601c (by the passenger side elevators)