



Writing JavaScript from a C++ background

Similarities with C++

- Mostly, JavaScript code looks like C++ code.



Similarities with C++

- Mostly, JavaScript code looks like C++ code.
- Flow control commands are still all the same:
`if/else/switch/for/while/do/break/continue`.
- Functions are still called with `()`, with input parameter values inside.
- Assignment still uses `=`



Similarities with C++

- Sequences of operations (statements) happen one after another, and can be separated by `;` or whitespace.
- Brackets `{ }` are still used to surround your subroutines, or if there are multiple statements you'd like in a code block or an `if / for / while / etc.`
- Boolean math and comparisons still look the same.

`&&` `||` `^=` `*=` `+=` `++` `&`



Similarities with C++

- Class definitions still look very familiar:

```
class dog
{
    constructor( name ) { this.name = name }
    bark() { }
}
```

```
new dog();
```



Similarities with C++

- Objects still have member variables and members are still obtained with . (dot).

```
poodle.bark();
```

```
this.times_barked = 9;           // Count barks
```

- "this" still refers to the current Object.
- Code comments are still // or /* */.



Main Difference from C++:



Main Difference from C++: Variable Declaration.

- JavaScript is a dynamically typed language
- It figures out type for you implicitly at run time
- Use “let” or “const” to declare every variable. Leave off the type.

Examples:

- A loop counter:

`let counter = 0` **vs** `int counter = 0`

- A custom matrix:

`let matrix = Mat4.identity()`




Main Difference from C++: Variable Declaration.

Types are left off in function arguments too; just separate the argument names with commas.

```
function draw( amount, color );
```

- You also don't have to warn JavaScript about return types. Just say “function” to declare a function.
- Return any expression you want when it's time to.

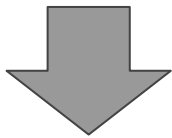
```
if(flag) return "a string"  
else return 9
```



More JavaScript Differences

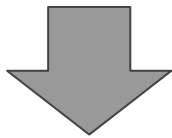
- “Arrow function” notation:
- You can use shorthand for declaring functions when you want.

```
function (x,y) { return x + y; }
```



```
(x,y) => x+y
```

```
function x { return x + 9 }
```



```
x => x+9
```



More JavaScript Differences


- Unlike C++, all semicolons are optional in JavaScript.
- Unlike Python, spacing and indentation are optional.
- Ints and floats are treated the same in JS -- no truncation errors.
 - Numbers are always stored as 64-bit floats.



More JavaScript Differences

- Functions can be declared right in the middle of whatever you're typing.
 - It's really convenient in JavaScript to pass functions around as callbacks to other functions.
 - In the below call, we pass in and define another function (a callback) for what the button should do.

```
make_button( "Go to world origin",  
  function( matrix ) { matrix.set_identity( 4,4 ) },  
  "orange" );
```



More JavaScript Differences

- Unlike C++, some things are buried in the `Math` namespace.
 - For trigonometry you say `Math.sin`, `Math.cos`, etc.
These expect radians.
 - More:
`Math.min`, `Math.pow`, `Math.max`, `Math.PI`,
`Math.log`, `Math.exp`



More JavaScript Differences

- The keyword `"this"` isn't automatically implied in JavaScript code (even if you're in a class member function!)
 - You have to fully spell out `this.draw_flower()` when calling your functions.
 - Or `this.animation_time` instead of just `animation_time` when accessing members.
 - You will forget.



More JavaScript Differences

- Javascript loses track of its `this` pointer quite easily
- Entering a function declared with `function` destroys it
- Entering an arrow function (`=>` shorthand) does not.



JavaScript Arrays

- Use square braces to build them anytime during an expression:

```
[ a, b, c ]
```

They behave like C++ `std::vector<>`'s, except they can combine multiple types of values.

```
let arr = [];
```

```
arr.push(9);
```

```
arr.push("some string");
```

```
let result = arr[0];
```



Array “Spread” Notation ...

- Suppose our array “a” has five items in it.
- Saying:

`...a`

- is identical to spelling out all five items with comma in between.
- This is really useful when a function wants lots of arguments.

```
let arr = [ "la", "la", "la" ];
```

```
arr2.push( ....arr );
```

```
// Same as: arr2.push( arr[0], arr[1],
```

```
//           arr[2] );
```



Array “Spread” Notation ...

- Another trick:
 - This expression generates a (shallow) copy of an array. Why?


```
[ ...arr ]
```



JavaScript Objects

- Javascript has a primitive data type called an object.
 - These are key-value lookup tables (like a dictionary)
 - Use curly braces to build them anytime in an expression.
 - Declare one yourself (a literal) like this:

```
{ "x": 1, "y": 2, "color": 3 }
```

- They are of type Object, **which is also everything's base class.**
 - They behave like C++ `std::map<string, >`'s, except they can combine multiple types of values.
- 

Debugging and Editing

- As with C++, following your code line by line using a **debugger tool** continues to be crucial for getting it to work.
 - If you've never used one, these let you watch each step of your code execute individually.
- You will rely on a debugger more than ever.
 - Default output of graphics? Blank screen.
 - Error state? Blank screen.



Debugging and Editing

- Note: In this class we'll assume **Google Chrome** as a code editor.
- Yes, it can actually do editing
 - It's found under developer tools.
- Getting it to work on local files requires some setup.
 - This setup is assignment 1.



Debugging and Editing

- Why are we assuming a particular text editor, and using Chrome of all things??
- It's nice because of it shares a user interface with Chrome's **Debugger**.
 - So you'll be more likely to use the **debugger** when your code doesn't work instead of coming to us.



Debugging and Editing

- What about other tools?
 - **VSCode** is a much better code editor than Chrome, and can also attach to Chrome's debugger
 - But for now let's just have one tool.
 - Firefox has a code editor too, called **WebIDE**.
 - Our advice and instructions only cover Chrome, due to limited time.



Debugging and Editing

- Surprisingly, all the debugging features from C++ debuggers like Visual Studio or XCode are there in the Chrome browser developer tools.
 - Pausing in between lines of code, to hover your mouse over each variable or expression and observe as they change



Debugging and Editing

- Chrome debugger compared to other debuggers (VS, XCode):
 - It has a console for typing arbitrary statements and seeing the result
 - If a line of code does multiple things, you can put a breakpoint on any of the parts
 - Visual Studio's most hyped feature, being able to drag around the instruction pointer, is just about all that's missing



Code comments in the 174a template

- We'll sometimes give you code skeletons that are full of instructive comments for understanding their process.
- The comments can not only help you learn to use the template, but also how a graphics program works.
- The code itself shows many JavaScript tricks you might want to mimic.



Code comments in the 174a template

```
class Vec extends Float32Array           // Vectors of floating point numbers.  This puts vector math into JavaScript.
| | | | | | | | | | | | | | | | | |   // See these examples for usage of each function:
//      equals: "Vec.of( 1,0,0 ).equals( Vec.of( 1,0,0 ) )" returns true.
//      plus: "Vec.of( 1,0,0 ).plus  ( Vec.of( 1,0,0 ) )" returns the Vec [ 2,0,0 ].
//      minus: "Vec.of( 1,0,0 ).minus ( Vec.of( 1,0,0 ) )" returns the Vec [ 0,0,0 ].
// mult-pairs: "Vec.of( 1,2,3 ).mult_pairs( Vec.of( 3,2,0 ) )" returns the Vec [ 3,4,0 ].
//      scale: "Vec.of( 1,2,3 ).scale( 2 )" overwrites the Vec with [ 2,4,6 ].
//      times: "Vec.of( 1,2,3 ).times( 2 )" returns the Vec [ 2,4,6 ].
// randomized: Returns this Vec plus a random vector of a given maximum length.
```

Stacks in Graphics

- Sometimes your code deals with values (especially math matrices) that change a lot of times
 - You may want them to outlive your function calls, resuming their previous value
- Because of that you will might want a "stack" to maintain the history of that value, for you to rewind as you please.



Stacks in Graphics

- All JavaScript arrays have stack functions
 - Making any array data member will look like:
this.history_stack = [];
 - Afterwards, **this.history_stack.push(some_matrix)** saves your current matrix
 - **this.history_stack.pop()** returns the most recent one (and takes it off the stack - if you don't want that, just read from the last array element)



Stacks in Graphics

- Old GPUs used to manage huge “stacks” for you (for history of your variables) because it is so common to design a scene as a hierarchy (like a tree).
- Modern GPUs have “programmable shaders” you make yourself instead of those stacks. We’ll learn about those soon.
 - We might not need any stacks.

