web222

WEB222 - Week 1

Preface

The web is the most ubiquitous computing platform in the world. As a developer, learning the web takes time. There are hundreds of languages, libraries, frameworks, and tools to be learned, some old, some built yesterday, and all being mixed together at once.

The fundamental unit of the web is the hyperlink–the web is interconnected. These weekly notes provide numerous links to external resources, books, blogs, and sample code. To get good at the web, you need to be curious and you need to go exploring, you need to try things.

Make sure you follow the links below as you read, and begin to create your own web of knowledge and experience. No one resource can begin to cover the breadth and depth of web development.

> Question: do I need to read the weekly notes? How about all the many links to external resources?

Yes, you do need to read the weekly notes. You will be tested on this material. We will discuss it in class, but not cover everything. The external links will help you understand and master the material. You are advised to read some external material, but you don't need to read all of it. However, make sure you *do* read Recommended Readings.

Internet Architecture

Overview

- How does the Internet work?
  - How the Internet works in 5 minutes (video)
- How the Web works

Application Protocols

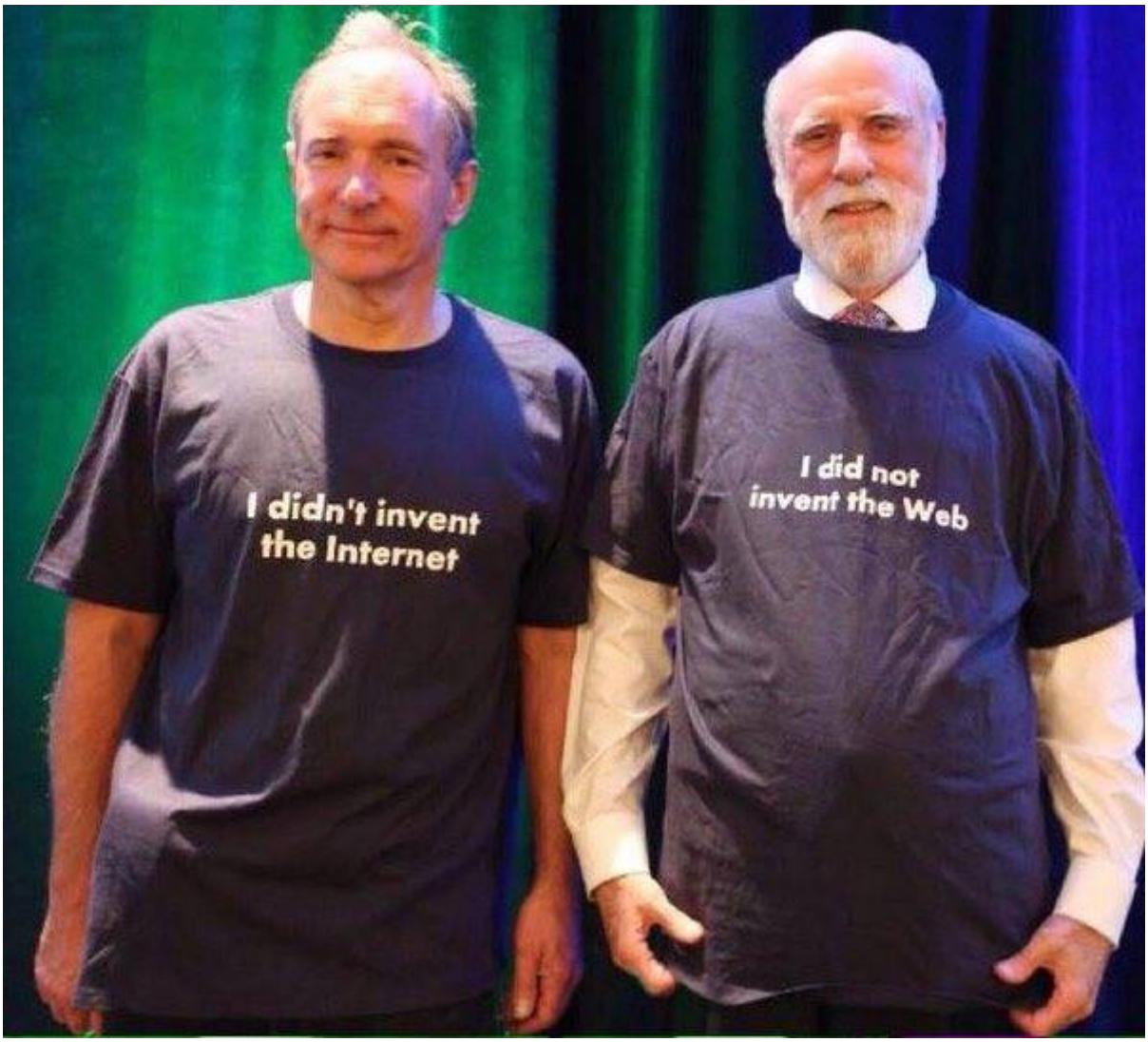
The web runs on-top of TCP/IP networks using a number of communication protocols, including:

- IP these 32-bit numbers (IPv4) are assigned to every device on the Internet (IPv6 uses 128-bit numbers).
- Domain Names human-readable addresses for servers on the Internet

- Domain Name System (DNS), the "Phone Book" of the Internet. There are many popular DNS servers you can use:
  - OpenDNS: `208.67.222.222`, `208.67.220.220`
  - Cloudeflare: `1.1.1.1`, `1.0.0.1`
  - Google: `8.8.8.8`, `8.8.4.4`
  - There are lots more, but each has trade offs (privacy, speed)
- Hypertext Transfer Protocol (HTTP)
  - How to get things on the web
  - HTTP Responses
- Hypertext Transfer Protocol Secure (HTTPS)

There are many more as well (SMTP, FTP, POP, IMAP, SSH, etc).

We often use the terms "Web" and "Internet" interchangeably, however, they aren't the same. Pictured below, Tim Berners-Lee (left), who invented the *World Wide Web*, and Vint Cert (right), who was one of the main inventors of the *Internet*:



The *World Wide Web* (WWW) runs on top of the Internet using HTTP, and allows us to access web services, request resources (i.e., pages, images), and transmit data between clients and servers. The web is a subset of the Internet.

The web isn't owned or controlled by any single company, organization, or government. Instead, it is defined as a set of open standards, which everyone building and using the web relies upon. Some examples of these standards include HTML, HTTP, SVG, and many more.

HTTP Requests and Responses

The Hypertext Transfer Protocol is a **stateless**, **client-server** model for formatting requests and responses between computers on the Internet. This means one computer makes a request (the client) to another (the server), and after the response is returned, the connection is closed.

The server listens for requests, and fulfills (or rejects) those requests by returning (or generating) the requested resources, such as images, web pages, videos, or other data.

URLs

Web resources are reachable via unique identifiers called a *Uniform Resource Locator* or *URL*. Consider the URL for this course's outline:

https://www.senecacollege.ca/cgi-bin/subject?s1=WEB222

A URL contains all the information necessary for a web client (e.g., a browser) to request the resource. In the URL given above we have:

- protocol: `https:` - the resource is available using the HTTPS (i.e., secure HTTP) protocol
- domain: `www.senecacollege.ca` - the domain (domain name) of the server. We could also have substituted the IP address (`23.208.15.99`), but it's easier to remember domain names.
- port: Not Given - if not specified, the port is the default for HTTP `80` or `443` for HTTPS. It could have been specified by appending `:443` like so: `https://www.senecacollege.ca:443`
- origin: combining the protocol, domain, and port gives us a unique origin, `https://www.senecacollege.ca`. Origins play a central role in the web's security model.
- path: `/cgi-bin/subject` - a filesystem-like path to the resource on the server. It may or may not end with a file extension (e.g., you might also have seen another server use `/cgi-bin/subject.html`)
- query string: `?s1=WEB222` - additional parameters sent to the server as part of the URL, of the form `name=value`

URLs can only contain a limited set of characters, and anything outside that set has to be *encoded*. This includes things like spaces, non-ASCII characters, Unicode, etc.

Requests

A URL describes the location (i.e., server, pathname) and how to interpret (i.e., which protocol) a resource on the Internet. To get the resource, we need to request it by sending a properly formatted HTTP Request to the appropriate server (host):

```
GET /cgi-bin/subject HTTP/1.1
Host: www.senecacollege.ca
```

Here we do a `GET` request using HTTP version 1.1 for the resource at the path `/cgi-bin/subject` on the server named `www.senecacollege.ca`.

There are various *HTTP Verbs* we can use other than `GET`, which allow us to request that resources be returned, created, deleted, updated, etc. The most common include:

- `GET` - retrieve the data at the given URL
- `POST` - create a new resource at the given URL based on the data sent along with the request in its *body*
- `PUT` - update an existing resource at the given URL with the data sent along with the request in its *body*
- `DELETE` - delete the resource at the given URL

We can use a URL in many ways, for example, via the command line using a tool like curl (NOTE: on Windows, use `curl.exe`):

```
$ curl https://www.senecacollege.ca/cgi-bin/subject?s1=WEB222

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML+RDFa 1.0//EN"
  "http://www.w3.org/MarkUp/DTD/xhtml-rdfa-1.dtd">
<html lang="en" dir="ltr"
  xmlns:content="http://purl.org/rss/1.0/modules/content/"
  xmlns:dc="http://purl.org/dc/terms/"
  xmlns:foaf="http://xmlns.com/foaf/0.1/"
  xmlns:og="http://ogp.me/ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:sioc="http://rdfs.org/sioc/ns#"
  xmlns:sioct="http://rdfs.org/sioc/types#"
  ...
</section> <!-- /.block -->
  </div>
</footer>
</body>
</html>
```

Responses

Upon receiving a request for a URL, the server will respond with an *HTTP Response*, which includes information about the response, and possibly the resource being requested. Let's use `curl` again, but this time ask that it `--include` the response headers:

```
$ curl --include https://www.senecacollege.ca/cgi-bin/subject\?s1\=WEB222
HTTP/1.1 200 OK
```

```
Content-Type: text/html;charset=ISO-8859-1
Strict-Transport-Security: max-age=16070400; includeSubDomains
Date: Wed, 06 Sep 2023 14:31:11 GMT
Content-Length: 17241
Connection: keep-alive

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml

<html xmlns="http://www.w3.org/1999/xhtml"><!-- InstanceBegin template="/Templates/mainTem
...
```

In this case, we see a two-part structure: first a set of **Response Headers**; then the actual HTML **Response Body**. The two are separated by a blank line. The headers provide extra metadata about the response, the resource being returned, the server, etc.

HTTP Headers are well defined, and easy to lookup via Google, MDN, or StackOverflow. They follow the `key: value` format, and can be upper- or lower-case:

`name: value`

For example, in the response above, we see a number of interesting things:

- `200 OK` - tells us that the requested resource was successful located and returned.
- Info about the `Date`
- The `Content-Type` is `text`, and more specifically, `html` (a web page) using ISO-8859-1 text encoding.

After these **headers** we have a blank line (i.e., `\n\n`), followed by the **body** of our response: the actual HTML document.

What if we requested a URL that we know doesn't exist?

```
$ curl --include https://ict.senecacollege.ca/course/web000

HTTP/1.1 302 Found
Date: Thu, 30 Aug 2018 20:25:28 GMT
Server: Apache/2.4.29 (Unix) OpenSSL/1.0.2l PHP/5.6.30
X-Powered-By: PHP/5.6.30
Expires: Sun, 19 Nov 1978 05:00:00 GMT
Cache-Control: no-cache, must-revalidate, post-check=0, pre-check=0
Location: https://ict.senecacollege.ca/Course/CourseNotFound?=web000
Content-Length: 0
Content-Type: text/html; charset=UTF-8
```

This time, instead of a `200` status code, we get `302`. This indicates that the resource has moved, and later in the headers we are given a new `Location` to try. Notice there is no body (not every response will include one).

Let's try following the suggested redirect URL:

```
curl -I https://www.senecacollege.ca/cgi-bin/subjec
HTTP/1.1 404 Not Found
Pragma: no-cache
Content-Type: text/html;charset=UTF-8
Strict-Transport-Security: max-age=16070400; includeSubDomains
Cache-Control: no-cache, no-store, must-revalidate
Date: Wed, 06 Sep 2023 14:35:04 GMT
Connection: keep-alive
Set-Cookie: JSESSIONID=25B152E3F14082DDB666AEE9BE3B4CA7; Path=/; Secure
Set-Cookie: TS0148d87b=01576d3f8972e643bf2b887be1dd52aa5ea6da872a01d2283959af6601471be59f4
...
```

Now a third response code has been returned, `404 Not Found` as well as another HTML page telling us our course couldn't be located.

There are dozens of response codes, but they fall into a few categories you should learn:

- `1xx` - information responses
- `2xx` – successful responses
- `3xx` - redirection messages
- `4xx` – client error responses
- `5xx` – server error responses

Web Browsers

So far we've been communicating with web servers using `curl`, but a more common tool is a **Web Browser**.

A good way to think about a browser is as an operating system vs. an application. A web browser provides implementations of the web's open standards. This means it knows how to communicate HTTP, DNS and other protocols over the network in order to request resources via URLs. It also contains parsers for the web's programming languages, and knows how to render, execute, and lay-out web content for use by a user. Browsers also contain lots of security features, and allow users to download and run untrusted code (i.e., code from a random server on the Internet), without fear of infecting their computers.

Some of the the largest software companies and vendors in the world all have their own browsers:

- Google Chrome for desktop and Android
- Microsoft Edge and Internet Explorer (IE)
- Apple Safari and Safari for iOS

- Mozilla Firefox
- Samsung Internet for Android
- Opera

There are hundreds more, and thousands of different OS and version combinations. There are good stats on usage info for desktop and mobile, but no one company or browser controls the entire web.

As a web developer, you can't ever know for sure which browser your users will have. This means you have to test your web applications in different browsers and on different platforms in order to make sure the experience is good for as many people as possible.

The web is also constantly evolving, as new standards are written, APIs and features added to the web platform, and older technologies retired. A good way to stay on top of what does and doesn't work in a particular browser is to use https://caniuse.com/. This is a service that keeps track of web platform features, and which browsers do and don't implement it.

For example, you can look at the `URL()` API, used to work with URLs in JavaScript: https://caniuse.com/#feat=url. Notice that it's widely supported (green) in most browsers (89.69% at the time of writing), but not supported (red) in some older browsers like Internet Explorer.

Because the web is so big, so complicated, so old, and used by so many people for so many different and competing things, it's common for things to break, for there to be bugs, and for you to have to adapt your code to work in interesting ways. The good news is, it means there are lots of jobs for web developers to make sure it all keeps working.

Uniqueness of the Web as a Platform

We've been discussing HTTP as a way to request URLs be transferred between clients and servers. The web is globally distributed set of

- services - requesting *data* (Text, JSON, XML, binary, etc) to be used in code (vs. looked at by a user)
- resources, pages, documents, images, media - both static and dynamic user viewable resources (web pages), which link to other similar resources.
- applications - a combination of the above, providing rich user interfaces for working with real-time data or other complex information, alone or in networked (i.e., collaborative) ways.

The web can be read-only. The web can also be interactive (video games), editable (wikis), personal (blog), and productive (e-commerce).

The web is *linkable*, which makes it something that can be indexed, searched, navigated, and connected. The web gets more valuable as its connections grow: just look at all the other pages and resources this page links to itself!

The web allows users to access and run remote applications *without* needing to install new software. **The deployment model of the web is HTTP**. Compare that to traditional software that has to be manually installed on every computer that needs to run it. The same is true with mobile phones and apps in the various app stores. On the web, updates get *installed* every time you open a URL.

> Question: how many mobile or desktop apps did you install today vs. how many websites did you visit?

The web works on *every* computing platform. You can access and use the web on desktop and mobile computers, on TVs and smartwatches, on Windows and Mac, in e-Readers and video game consoles. The web works everywhere, and learning how to develop software for the web extends your reach into all those platforms.

Front-End Web Development: HTML5, CSS, JavaScript, and friends

When we talk about programming for the web in a browser, we often refer to this as *Front-End Web Development*. This is in contrast to server-side, or *Back-End Development*. In this course we will be focused on the front-end, leaving back-end for subsequent courses.

The modern web, and modern web browsers, are incredibly powerful. What was once possible only on native operating systems can now be done within browsers using only web technologies (cf. running Windows 2000 or Doom 3 in a browser window!)

The set of front-end technologies that make this possible, and are commonly referred to as the Web Platform, include:

- HTML5 - the Hypertext Markup Language, and its associated APIs, provide a way to define and structure content
- CSS - Cascading Style Sheets allow developers and designers to create beautiful and functional UIs for the web
- JS - JavaScript allows complex user interaction with web content, and dynamic behaviours in documents and applications.
- DOM - the Document Object Model and its APIs allows scripts and web content to interact at runtime.
- Web APIs - hundreds of APIs provide access to hardware devices, networking, files, 2D and 3D graphics, databases, and so much more.
- WebAssembly or WASM - a low-level assembly language that can be run in web browsers, allowing code written in C/C++ and other non-web languages to target the web. For example, Google Earth uses WebAssembly.

In addition to these primary technologies, an increasingly important set of secondary, or third-party technologies are also in play:

- Libraries, Modules - Bootstrap, Leaflet, Three.js, Lodash, ...

- Frameworks - React, Angular, Vue.js, ...
- Tooling - Babel, webpack, ESLint, Prettier
- Languages that "compile" to JavaScript - because JavaScript runs everywhere, many languages target the web by "compiling" (also know as *transpiling*) to JavaScript. A good example is TypeScript.

The front-end web stack is also increasingly being used to build software outside the browser, both on desktop and mobile using things like Electron and Progressive Web Apps (PWA). Visual Studio Code, for example, is written using web technologies and runs on Electron, which is one of the reasons it works across so many platforms. You can also run it entirely in the browser: vscode.dev.

Introduction to JavaScript

The first front-end web technology we will learn is JavaScript. JavaScript (often shortened to JS) is a lightweight, interpreted or JIT (i.e., Just In Time) compiled language meant to be embedded in host environments, for example, web browsers.

JavaScript looks similar to C/C++ or Java in some of its syntax, but is quite different in philosophy; it is more closely related to Scheme than C. For example, JavaScript is a dynamic scripting language supporting multiple programming styles, from object-oriented to imperative to functional.

JavaScript is one of, if not the most popular programming languages in the world, and has been for many years. Learning JavaScript well will be a tremendous asset to any software developer, since so much of the software we use is built using JS.

> JavaScript's many versions: JavaScript is an evolving language, and you'll hear it referred to by a number of names, including: ECMAScript (or ES), ES5, ES6, ES2015, ES2017, ..., ES2021, ES2022, etc. ECMA is the European Computer Manufacturers Association, which is the standards body responsible for the JS language. As the standard evolves, the specification goes through different versions, adding or changing features and syntax. In this course we will primarily focus on ECMAScript 6 (ES6) and newer versions, which all browsers support. We will also sometimes use new features of the language, which most browsers support. Language feature support across browsers is maintained in this table.

JavaScript Resources

Throughout the coming weeks, we'll make use of a number of important online resources. They are listed here so you can make yourself aware of them, and begin to explore on your own. All programmers, no matter how experienced, have to return to these documents on a routine basis, so it's good to know about them.

- JavaScript on MDN

    - JavaScript Guide
    - JavaScript Reference
- Eloquent JavaScript
- JavaScript for impatient programmers (ES2022 edition)

JavaScript Environments

Unlike C, which is compiled to machine code, JavaScript is meant to be run within a host environment. There are many possible environments, but we will focus on the following:

- Web Browsers, and their associated developer tools, primarily:
    - Chrome DevTools
    - Firefox Developer Tools
- node.js, and its command line REPL (Read-Eval-Print-Loop)

If you haven't done so already, you should install all of the above.

JavaScript Engines

JavaScript is parsed, executed, and managed (i.e., memory, garbage collection, etc) by an engine written in C/C++. There are a number of JavaScript engines available, the most common of which are:

- V8, maintained an used by Google in Chrome and in node.js
- SpiderMonkey, maintained and used by Mozilla in Firefox
- ChakraCore, maintained and used by Microsoft in Edge
- JavaScriptCore, maintained and used by Apple in Safari

These engines, much like car engines, are meant to be used within a larger context. We will encounter them indirectly via web browsers and in node.js.

It's not important to understand a lot about each of these engines at this point, other than to be aware that each has its own implementation of the ECMAScript standards, its own performance characteristics (i.e., some are faster at certain things), as well as its own set of bugs.

Running JavaScript Programs

JavaScript statements can be stored in an external file with a `.js` file extension, or embedded within HTML code via the HTML `<script>` element. As a developer, you also have a number of options for writing and executing JavaScript statements or files:

1. From the command line via node.js. You'll learn more about node.js in subsequent courses, but we'll also use it sometimes in this course to quickly try test JavaScript expressions, and to run JavaScript programs outside the browser.

2. Using Firefox's Developer Tools, and in particular the Web Console, JavaScript Debugger, and Scratchpad.
3. Using Chrome's DevTools, and in particular the Console and Sources Debugger
4. Finally, we'll eventually write JavaScript that connects with HTML and CSS to create dynamic web pages and applications.

Take some time to install and familiarize yourself with all of the methods listed above.

JavaScript Syntax

Recommend Readings

We will spend a month learning JavaScript, and there is no one best way to do it. The more you read and experiment the better. The following chapters/pages give a good overview:

- Chapter 1. Basic JavaScript of Exploring JS (ES5).
- MDN JavaScript Introduction Tutorial
- Chapter 1. Values, Types and Operators and Chapter 2. Program Structure of Eloquent JavaScript (2nd Ed.).

Important Ideas

- Like C, JavaScript is Case-Sensitive: `customerCount` is not the same thing as `CustomerCount` or `customercount`
- Name things using `camelCase` (first letter lowercase, subsequent words start with uppercase) vs. `snake_case`.
- Semicolons are optional in JavaScript, but highly recommended. We'll expect you to use them in this course, and using them will make working in C++, Java, CSS, etc. much easier, since you have to use them there.
- Comments work like C/C++, and can be single or multi-line

```
// This is a single line comment. NOTE: the space between the // and first letter.

/*
 This is a multi-line comment,
 and can be as long as you need.
 */
```

- Whitespace: JavaScript will mostly ignore whitespace (spaces, tabs, newlines). In this course we will expect you to use good indentation practices, and for your code to be clean and readable. Many web programmers use Prettier to automatically format their code, and we will too:

```
// This is poorly indented, and needs more whitespace
function add(a,b ){
if(!b){
        return a;
}else {
return a+b;
}}

// This is much more readable due to the use of whitespace
function add(a, b) {
    if(!b) {
        return a;
    } else {
        return a + b;
    }
}
```

- JavaScript statements: a JavaScript program typically consists of a series of statements. A statement is a single-line of instruction made up of objects, expressions, variables, and events/event handlers.
- Block statement: a block statement, or compound statement, is a group of statements that are treated as a single entity and are grouped within curly brackets `{...}`. Opening and closing braces need to work in pairs. For example, if you use the left brace `{` to indicate the start of a block, then you must use the right brace `}` to end it. The same matching pairs applies to single `'......'` and double `"......."` quotes to designate text strings.
- Functions are one of the primary building blocks of JavaScript. A function defines a subprogram that can be called by other parts of your code. JavaScript treats functions like other built-in types, and they can be stored in variables passed to functions, returned from functions or generated at run-time. Learning how to write code in terms of functions will be one of your primary goals as you get used to JavaScript.
- Variables are declared using the `let` keyword. You must use the `let` keyword to precede a variable name, but you do not need to provide a type, since the initial value will set the type.

> JavaScript version note: JavaScript also supports the `var` and `const` keywords for variable declaration. We will primarily use `let` in this course, but be aware of `var` and `const` as well, which other developers will use.

```
let year;
let seasonName = "Fall";

// Referring to and using syntax:
year = 2023;
console.log(seasonName, year);
```

- JavaScript Variables: variables must start with a letter (`a-zA-z`), underscore (`_`), or dollar sign (`$`). They cannot be a reserved (key) word. Subsequent characters can be letters, numbers, underscores.

NOTE: If you forget to use the `let` keyword, JavaScript will still allow you to use a variable, and simply create a *global variable*. We often refer to this as "leaking a global," and it should always be avoided:

```
let a = 6;      // GOOD: a is declared with let
b = 7;          // BAD: b is used without declaration, and is now a global
```

- Data Types: JavaScript is a typeless language–you don't need to specify a type for your data (it will be inferred at runtime). However, internally, the following data types are used:
  - `Number` - a double-precision 64-bit floating point number. Using `Number` you can work with both Integers and Floats. There are also some special `Number` types, `Infinity` and `NaN` .
  - `BigInt` - a value that can be too large to be represented by a `Number` (larger than `Number. MAX_SAFE_INTEGER` ,) can be represented by a `BigInt` . This can easily be done by appending `n` to the end of an integer value.
  - `String` - a sequence of Unicode characters. JavaScript supports both single (`'...'`) and double (`"..."`) quotes when defining a `String` .
  - `Boolean` - a value of `true` or `false` . We'll also see how JavaScript supports so-called *truthy* and *falsy* values that are not pure `Boolean` s.
  - `Object` , which includes `Function` , `Array` , `Date` , and many more. - JavaScript supports object-oriented programming, and uses objects and functions as first-class members of the language.
  - `Symbol` - a primitive type in JavaScript that represents a unique and anonymous value/identifier. They can normally be used as an object's unique properties.
  - `null` - a value that means "this is intentionally nothing" vs. `undefined`
  - `undefined` - a special value that indicates a value has never been defined.

Declaration	Type	Value
`let s1 = "some text";`	`String`	`"some text"`
`let s2 = 'some text';`	`String`	`"some text"`
`let s3 = '172';`	`String`	`"172"`
`let s4 = '172' + 4;`	`String`	`"1724"` (concatenation vs. addition)
`let n1 = 172;`	`Number`	`172` (integer)
`let n2 = 172.45;`	`Number`	`172.45` (double-precision float)

Declaration	Type	Value
`let n3 = 9007199254740993n;`	`BigInt`	`9007199254740993n` (integer)
`let b1 = true;`	`Boolean`	`true`
`let b2 = false;`	`Boolean`	`false`
`let b3 = !b2;`	`Boolean`	`true`
`let s = Symbol("Sym");`	`symbol`	`Symbol(Sym)`
`let c;`	`undefined`	`undefined`
`let d = null;`	`object`	`null`

Consider a simple program from your C course, and how it would look in JavaScript

```c
// Area of a Circle, based on https://scs.senecac.on.ca/~btp100/pages/content/input.html
// area.c

#include <stdio.h>               // for printf

int main(void)
{
   const float pi = 3.14159f;   // pi is a constant float
   float radius = 4.2;          // radius is a float
   float area;                  // area is a float

   area = pi * radius * radius; // calculate area from radius

   printf("Area = %f\n", area); // copy area to standard output

   return 0;
}
```

Now the same program in JavaScript:

```javascript
const pi = 3.14159;                     // pi is a Number
let radius = 4.2;                       // radius is a Number
let area;                               // area is (currently) undefined

area = pi * radius * radius;            // calculate area from radius

console.log("Area = " + area );         // print area to the console
```

We could also have written it like this, using `Math.PI`, which we'll learn about later:

```
let radius = 4.2;                       // radius is a Number
let area = Math.PI * radius * radius;   // calculate area from radius

console.log("Area", area);              // print area to the console
```

- Common JavaScript Operators (there are more, but these are a good start):

Operator	Operation	Example
`+`	Addition of `Number`s	`3 + 4`
`+`	Concatenation of `String`s	`"Hello " + "World"`
`-`	Subtraction of `Number`s	`x - y`
`*`	Multiplication of `Number`s	`3 * n`
`/`	Division of `Number`s	`2 / 4`
`%`	Modulo	`7 % 3` (gives 1 remainder)
`++`	Post/Pre Increment	`x++`, `++x`
`--`	Post/Pre Decrement	`x--`, `--x`
`=`	Assignment	`a = 6`
`+=`	Assignment with addition	`a += 7` same as `a = a + 7`. Can be used to join `String`s too
`-=`	Assignment with subtraction	`a -= 7` same as `a = a - 7`
`*=`	Assignment with multiplication	`a *= 7` same as `a = a * 7`
`/=`	Assignment with division	`a /= 7` same as `a = a / 7`
`&&`	Logical `AND`	`if(x > 3 && x < 10)` both must be `true`
`()`	Call/Create	`()` invokes a function, `f()` means invoke/call function stored in variable `f`
`||`	Logical `OR`	`if(x === 3 || x === 10)` only one must be `true`
`|`	Bitwise `OR`	`3.1345|0` gives `3` as an integer

Operator	Operation	Example
`!`	Logical `NOT`	`if(!(x === 2))` negates an expression
`==`	Equal	`1 == 1` but also `1 == "1"` due to type coercion
`===`	Strict Equal	`1 === 1` but `1 === "1"` is not `true` due to types. Prefer `===`
`!=`	Not Equal	`1 != 2`, with type coercion
`!==`	Strict Not Equal	`1 !== "1"`. Prefer `!==`
`>`	Greater Than	`7 > 3`
`>=`	Greater Than Or Equal	`7 >=7` and `7 >= 3`
`<`	Less Than	`3 < 10`
`<=`	Less Than Or Equal	`3 < 10` and `3 <=3`
`typeof`	Type Of	`typeof "Hello"` gives `'string'`, `typeof 6` gives `'number'`
`cond ? a : b`	Ternary	`status = (age >= 18) ? 'adult' : 'minor';`

> JavaScript version note: you may encounter `=>` in JavaScript code, which looks very similar to `<=` or `>=`. If you see `=>` it is an arrow function, which is new ES6 syntax for declaring a function expression. We will slowly introduce this syntax, especially in later courses.

- JavaScript is dynamic, and variables can change value *and* type at runtime:

```
let a;              // undefined
a = 6;              // 6, Number
a++;                // 7, Number
a--;                // 6, Number
a += 3;             // 9, Number
a = "Value=" + a;   // "Value=9", String
a = !!a;            // true, Boolean
a = null;           // null
```

- JavaScript is a garbage collected language. Unlike C, memory automatically gets freed at runtime when variables are not longer in scope or reachable. We still need to be careful not to leak memory (i.e., hold onto data longer than necessary, or forever) and block the garbage collector from doing its job.
- Strings: JavaScript doesn't distinguish between a single `char` and a multi-character `String`–everything is a `String`. You define a `String` using either single (`'...'`), double

(`"..."`) quotes. Try to pick one style and stick with it within a given file/program vs. mixing them.

- JavaScript version note: newer versions of ECMAScript also allow for the use of template literals. Instead of `'` or `"`, a template literal uses ` (backticks), and you can also interpolate expressions.

- A JavaScript expression is any code (e.g., literals, variables, operators, and expressions) that evaluates to a single value. The value may be a `Number`, `String`, an `Object`, or a logical value.

```
let a = 10 / 2;                       // arithmetic expression
let b = !(10 / 2);                    // logical expression evaluates to false
let c = "10 " + "/" + " 2";           // string, evaluates to "10 / 2"
let f = function() { return 10 / 2;}; // function expression, f can now be called via the
let d = f();                          // f() evaluates to 10/2, or the Number 5
```

- JavaScript execution flow is determined using the following four (4) basic control structures:
  - Sequential: an instruction is executed when the previous one is finished.
  - Conditional: a logical condition is used to determine which instruction will be executed next - similar to the `if` and `switch` statements in C (which JavaScript also has).
  - Looping: a series of instructions are repeatedly executed until some condition is satisfied - similar to the `for` and `while` statements in C (which JavaScript also has). There are many different types of loops in JavaScript: for example `for` loops and `while` loops, as well as ways to `break` out of loops or skip iterations with `continue`. We'll cover other types as we learn about `Object` and `Array`.
  - Transfer: jump to, or invoke a different part of the code - similar to calling a function in C.

```
/**
 * 1. Sequence example: each statement is executed one after the other
 **/
let a = 3;
let b = 6;
let c = a + b;


/**
 * 2. Conditional examples: a decision is made based on the evaluation of an expression,
 * and a code path (or branch) taken.
 **/
let grade;
let mark = 86;

if (mark >= 90) {
```

```
    grade = 'A+';
} else if (mark >= 80) {
    grade = 'A';
} else if (mark >= 70) {
    grade = 'B';
} else if (mark >= 60) {
    grade = 'C';
} else if (mark >= 50) {
    grade = 'D';
} else {
    grade='F';
}

switch(grade) {
    case 'A+':
        // do these steps if grade is A+
        break;
    case 'A':
        // do these steps if grade is A
        break;
    case 'B':
        // do these steps if grade is B
        break;
    case 'C':
        // do these steps if grade is C
        break;
    case 'D':
        // do these steps if grade is D
        break;
    default:
        // do these steps in any other case
        break;
}


/**
 * 3. Looping example: a set of statements are repeated
 **/

let total = 0;
for(let i = 1; i < 11; i++) {
    total += i;
    console.log("i", i, "total", total);
}


/**
 * 4. Transfer example: a set of statements are repeated
 **/

function add(a, b) {        // declaring the add function
    if(!b) {                // check if the b argument exists/has a value
        return a;           // if not, simply return the value of argument a
```

```
    }
    return a + b;           // otherwise, return the two arguments added together
}

let total;
total = add(56);            // invoking the add function with a single argument
total = add(total, 92);     // invoking the add function with two arguments
```

Practice Exercises

Try to solve each of the following using JavaScript. If you need to `print` something, use `console.log()`, which will print the argument(s) you give it.

1. Create a variable `label` and assign it the value `"senecacollege"`. Create another variable `tld` and assign it `"ca"`. Create a third variable `domainName` that combines `label` and `tld` to produce the value `"senecacollege.ca"`.
2. Create a variable `isSeneca` and assign it a boolean value (`true` or `false`) depending on whether or not `domainName` is equal to `"senecacollege.ca"`. HINT: use `===` and don't write `true` or `false` directly.
3. Create a variable `isNotSeneca` and assign it the inverse boolean value of `isSeneca`. HINT: if `isSeneca` is `true`, `isNotSeneca` should be `false`.
4. Create four variables `byte1`, `byte2`, `byte3`, `byte4`, and assign each of these a value in the range `0-255`.
5. Convert `byte1` to a `String` using `.toString()`, and `console.log()` the result. What happens if you use `toString(2)` or `toString(16)` instead?
6. Create a variable `ipAddress` and assign it the value of combining your four `byteN` variables together, separated by `"."`. For example: `"192.168.2.1"`.
7. Create a variable `ipInt` and assign it the integer value of bit-shifting (`<<`) and adding your `byteN` variables. HINT: your `ipInt` will contain 32 bits, the first byte needs to be shifted 24 bit positions (`<< 24`) so it occupies 32-25, the second shifted 16, the third 8.
8. Create a variable `ipBinary` that contains the binary representation of the `ipInt` value. HINT: use `.toString(2)` to display the number with `1` and `0` only.
9. Create a variable `statusCode`, and assign it the value for the "I'm a teapot" HTTP status code. HINT: see https://developer.mozilla.org/en-US/docs/Web/HTTP/Status
10. Write an `If` statement that checks to see if your `statusCode` is a `4xx` client error. HINT: use the `<`, `>`, `>=`, and/or `<=` operators to test the value
11. Write a `switch` statement that checks your `statusCode` for all possible `1xx` information responses. In each case, you should `console.log()` the response text associated with the status code, or `"unknown information response"` if the status code is not known.
12. Write a function `is2xx(status)` which takes a status code `status` (e.g., `200`) and returns `true` if the status code is a valid 2xx code.

13. Create a variable `studentName` and assign your name. Create another variable `studentAge` and assign it your age. Use `console.log()` to print out a sentence that includes both variables, like `"Alice is 20 years old."`.
14. Create a variable `isEven` and assign it a boolean value (`true` or `false`) depending on whether a given number `num` is even or not. HINT: use the modulus operator `%`.
15. Create a variable `isOdd` and assign it the inverse boolean value of `isEven`. HINT: if `isEven` is `true`, `isOdd` should be `false`.
16. Create a variable `radius` and assign it a value of `10`. Calculate the area of a circle with this radius and assign the result to a variable `area`. HINT: use `Math.PI` and the formula `area = πr^2`.
17. Create a variable `temperatureInCelsius` and assign it a value. Convert this temperature to Fahrenheit and assign the result to a variable `temperatureInFahrenheit`. HINT: use the formula `F = C * 9/5 + 32`.
18. Create a variable `heightInFeet` and assign it a value. Convert this height to meters and assign the result to a variable `heightInMeters`. HINT: use the conversion factor `1 foot = 0.3048 meters`.
19. Create a variable `seconds` and assign it a value. Convert this time to minutes and seconds (e.g., 90 seconds becomes 1 minute and 30 seconds) and assign the result to two variables `minutes` and `remainingSeconds`.
20. Create a variable `score` and assign it a value. Write an `if` statement that checks if the score is an A (90-100), B (80-89), C (70-79), D (60-69), or F (below 60) and assigns the result to a variable `grade`.
21. Write a `switch` statement that checks the value of a variable `day` and `console.log()`s whether it is a weekday or weekend. HINT: `day` can be a value from 1 (Monday) to 7 (Sunday).
22. Write a function `isPositive(num)` which takes a number `num` and returns `true` if the number is positive and `false` otherwise.
23. Write a function `isLeapYear(year)` which takes a year `year` and returns `true` if the year is a leap year and `false` otherwise. HINT: a leap year is divisible by 4, but not by 100, unless it is also divisible by 400.
24. Write a function `getDayOfWeek(day)` which takes a number `day` (from 1 to 7) and returns the day of the week as a string (e.g., “Monday”).
25. Write a function `getFullName(firstName, lastName)` which takes two strings `firstName` and `lastName` and returns the full name as a single string.
26. Write a function `getCircleArea(radius)` which takes a number `radius` and returns the area of a circle with that radius.
27. Write a function `getHypotenuse(a, b)` which takes two numbers `a` and `b` (the lengths of the two sides of a right triangle) and returns the length of the hypotenuse. HINT: use the Pythagorean theorem and `Math.sqrt()` to calculate the square root.

After you try writing these yourself, take a look at a possible solution.

This site is open source. Improve this page.