COMP1531

🛠️ Development - Javascript

Lecture 1.4

Author(s): Hayden Smith

(Download as PDF)

In This Lecture

- **Why? 🤔**
  - Javascript is a valuable tool to learn and necessary for the project
- **What? 📰**
  - Learning a second language
  - Javascript vs C
  - Core javascript language features

👩🏻 Disclaimer

Because you already know C, we will teach Javascript very quickly and mainly focus on the differences between Javascript and C.

Unlike C, Javascript has a sprawling set of capabilities - the language will feel much bigger, and therefore you might feel you have a poorer grasp on it.

Don't expect to know everything about Javascript this term. Just focus on only learning what you need to solve a problem at hand, and you will learn more super quick.

🤟 Javascript

Javascript is a high level multi-paradigm scripting language with a massive popularity in the web-based software engineering space.

Javascript has become the universal go-to language to build web-based applications, which is more and more becoming the primary way we consume end-user applications

🤟 Javascript

Javascript is a high level multi-paradigm scripting language with a massive popularity in the web-based software engineering space.

Javascript has become the universal go-to language to build web-based applications, which is more and more becoming the primary way we consume end-user applications

```
1 const z = 3;
2 function hello(a, b, c) {
3   return `${a} ${b}`;
4 }
```

1.4_intro.js

🤟🏽 Javascript

Why?

- Javascript has an extremely rich open source library and package manager that allows you to build apps quickly.
- Javascript is very high level, making it easy to write code.
- Javascript is very well supported in industry, and that support is increasing.
- Javascript is the foundational language for a large and increasing number of software engineering companies (largely due to the saturation of web-based products). Javascript is the langauge of the web.

Learning Another Language

Learning another programming language is a very doable exercise, particularly if the language is from the same programming paradigm. Let's compare some languages.

	Procedural	Object-oriented	Typed	Pointers	Compiled
C	Yes	No	Yes	Yes	Yes
C++	Yes	Yes	Yes	Yes	Yes
Java	No	Yes	Yes	No	Yes
Python	Yes	Yes	Can be	No	No
Javascript	Yes	Yes	Can be	No	No

Once you know a language from a paradigm, others are much easier.

Learning Another Language

In the case of learning another language like Javascript after doing COMP1511 with C, the main hurdles we have to overcome are:

- Javascript does not have programmer-defined types, unlike C
- Javascript has object-oriented components (which we can somewhat ignore), unlike C
- Javascript does not deal with pointers, unlike C (yay)
- Javascript is often written at a "higher level" (more abstract)
- Javascript does not have an intermediate compilation step, like C

 Javascript VS C

Write a function that takes in two numbers, and returns the smaller number

C

```c
int minimum(int a, int b) {
  if (a > b) {
    return b;
  } else {
    return a;
  }
}
```

1.4_compare_1.c

Javascript

```javascript
function minimum(a, b) {
  if (a > b) {
    return b;
  } else {
    return a;
  }
}
```

1.4_compare_1.js

 Javascript VS C

Write a function that takes in two numbers, and returns the smaller number

C

```c
int minimum(int a, int b) {
  if (a > b) {
    return b;
  } else {
    return a;
  }
}
```

1.4_compare_1.c

Javascript

```javascript
function minimum(a, b) {
  if (a > b) {
    return b;
  } else {
    return a;
  }
}
```

1.4_compare_1.js

Now let's call the function and print the result!

Javascript VS C

Write a function that takes in two numbers, and returns the smaller number

C

```c
#include <stdio.h>

int minimum(int a, int b) {
  if (a > b) {
    return b;
  } else {
    return a;
  }
}

int main(int argc, char* argv[]) {
  printf("%d\n", minimum(3, 5));
}
```

1.4_compare_2.c

Javascript

```javascript
function minimum(a, b) {
  if (a > b) {
    return b;
  } else {
    return a;
  }
}

console.log(minimum(3, 5));
```

1.4_compare_2.js

Javascript VS C

Write a function that takes in two numbers, and returns the smaller number

C

```c
#include <stdio.h>

int minimum(int a, int b) {
  if (a > b) {
    return b;
  } else {
    return a;
  }
}

int main(int argc, char* argv[]) {
  printf("%d\n", minimum(3, 5));
}
```

1.4_compare_2.c

Javascript

```javascript
function minimum(a, b) {
  if (a > b) {
    return b;
  } else {
    return a;
  }
}

console.log(minimum(3, 5));
```

1.4_compare_2.js

Now let's run the program

Javascript VS C

C

```c
#include <stdio.h>

int minimum(int a, int b) {
  if (a > b) {
    return b;
  } else {
    return a;
  }
}

int main(int argc, char* argv[]) {
  printf("%d\n", minimum(3, 5));
}
```

1.4_compare_2.c

```
gcc -Wall -Werror -O -o 1.4_compare_2.c -o runnable
./runnable
```

Javascript

```javascript
function minimum(a, b) {
  if (a > b) {
    return b;
  } else {
    return a;
  }
}

console.log(minimum(3, 5));
```

1.4_compare_2.js

```
node 1.4_compare_2.js
```

Javascript VS C

C

```c
#include <stdio.h>

int minimum(int a, int b) {
  if (a > b) {
    return b;
  } else {
    return a;
  }
}

int main(int argc, char* argv[]) {
  printf("%d\n", minimum(3, 5));
}
```

1.4_compare_2.c

```
gcc -Wall -Werror -O -o 1.4_compare_2.c -o runnable
./runnable
```

Javascript

```javascript
function minimum(a, b) {
  if (a > b) {
    return b;
  } else {
    return a;
  }
}

console.log(minimum(3, 5));
```

1.4_compare_2.js

```
node 1.4_compare_2.js
```

OK but:

 Javascript VS C

C

```c
#include <stdio.h>

int minimum(int a, int b) {
  if (a > b) {
    return b;
  } else {
    return a;
  }
}

int main(int argc, char* argv[]) {
  printf("%d\n", minimum(3, 5));
}
```

1.4_compare_2.c

```
gcc -Wall -Werror -O -o 1.4_compare_2.c -o runnable
./runnable
```

Javascript

```javascript
function minimum(a, b) {
  if (a > b) {
    return b;
  } else {
    return a;
  }
}

console.log(minimum(3, 5));
```

1.4_compare_2.js

```
node 1.4_compare_2.js
```

OK but:

What is node? 😵

🤺 Javascript VS C

C

```c
#include <stdio.h>

int minimum(int a, int b) {
  if (a > b) {
    return b;
  } else {
    return a;
  }
}

int main(int argc, char* argv[]) {
  printf("%d\n", minimum(3, 5));
}
```

1.4_compare_2.c

```
gcc -Wall -Werror -O -o 1.4_compare_2.c -o runnable
./runnable
```

Javascript

```javascript
function minimum(a, b) {
  if (a > b) {
    return b;
  } else {
    return a;
  }
}

console.log(minimum(3, 5));
```

1.4_compare_2.js

```
node 1.4_compare_2.js
```

OK but:

What is node? 😵

Are there steps missing? 😰

🥳 NodeJS

NodeJS is a command line interface that interprets Javascript code within a runtime environment that is built on Google's V8 engine. 😵‍💫

Or if you want a simpler explanation...

NodeJS is the program that compiles and runs Javascript.

To really oversimplify it, NodeJS has a similar function to GCC.

🥳 NodeJS

NodeJS is what's known as an **interpreted** language instead of a **compiled** language.

This means that the program is compiled and run as part of the same *step*.

This has two implication:

- A little slower to run, because it has to compile to runnable code every time.
- A little more convenient, as changes to code don't require an extra compilation step.

🤺 Performance V Convenience

But let's go and learn more about the language...

Variables, Printing

const,let,console.log

```
// Variables declared with "let"
//  can be modified after definition
const years = 5;

// Variables declared with "const"
//  cannot be modified after definition
const name = 'Giraffe';
const age = 18;
const height = 2048.11;
const notexist = undefined;
const existbutnothing = null;

// You print with console.log
console.log(years);
console.log(name);
console.log(height);

// Double and single apostrophes are equivalent
console.log('Hello!');
console.log('how are you?');
```

1.4_variables.js

🦄 Strings

Concatenation, string literals

```
1  // We can easily join strings together!
2  let sentence = 'My';
3  sentence = sentence + ' name is';
4  sentence += ' Pikachu';
5  console.log(sentence);
6
7  // If you need to mix variables and
8  //  strings, you can create a string literal
9  const age = 7;
10 const name = 'Hayden';
11 const phrase = `Hello! My name is ${name} and I am ${age}`;
12 console.log(phrase);
```

1.4_strings.js

Also come single & double apostrophe.

Control Structures

`if`, `else if`, `else`, `while`, `for`.

```
1  const number = 5;
2  if (number > 10) {
3    console.log('Bigger than 10');
4  } else if (number < 2) {
5    // Do nothing
6  } else {
7    console.log('Number between 2 and 9');
8  }
9
10 console.log('--------------------------');
11
12 let i = 0;
13 while (i < 5) {
14   console.log('Hello there');
15   i += 1;
16 }
17
18 console.log('--------------------------');
19
20 for (let i = 0; i < 5; i++) {
21   console.log('Hello there');
22 }
```

1.4_control_structures.js

Functions

Very similar syntax to C

```
1 function minimum(a, b) {
2   if (a > b) {
3     return b;
4   } else {
5     return a;
6   }
7 }
```

1.4_compare_1.js

Pause for a bit of theory...

 Data Structures: Collections

We'll now discuss two important data structures that are both **collections** of data.

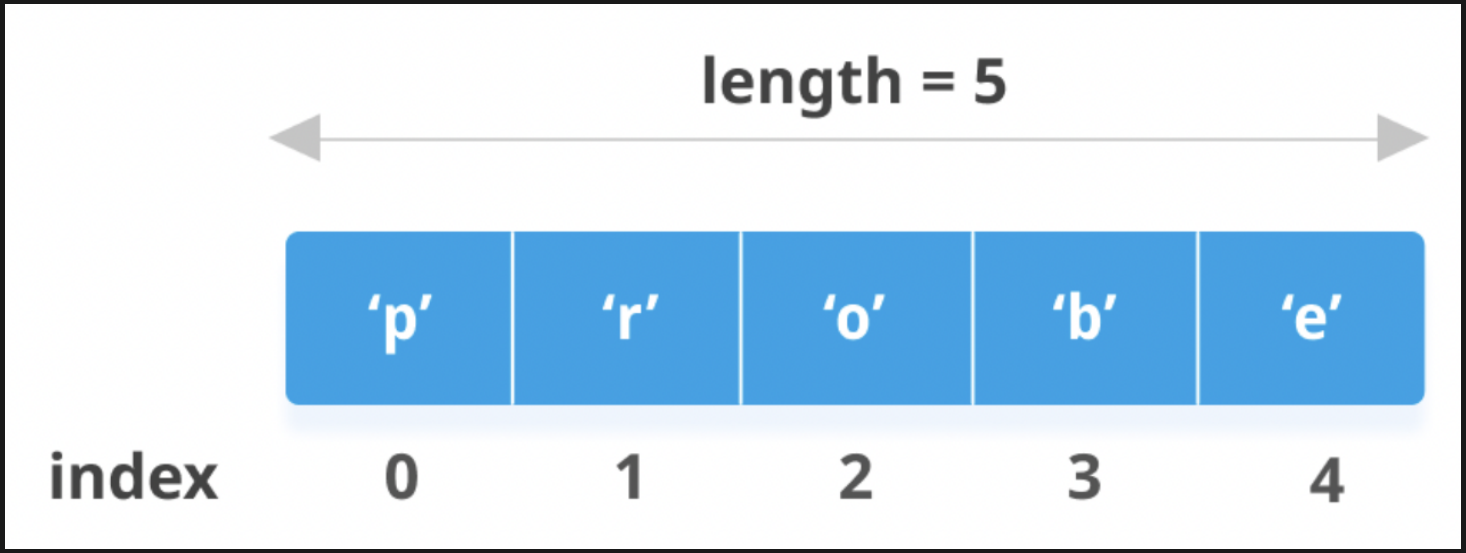
Collections can either be:

- Sequential collections
- Associative collections

🔢 Sequential Collections

In **sequential collections** values are referenced by their integer index (key) that represents their location in an order.

In Javascript sequential collections are represented by an **array**. In Javascript, arrays are used for both C-style arrays and C-style linked lists.



 Associative Collections

In **associative collections** values are referenced by their string key that maps to a value. They often do not have an inherent sense of order.

They're kind of like C `structs`, except the structure does not have to be defined at compile time.

- `name` → `"sally"`
- `age` → `18`
- `height` → `"187cm"`

Unpause, back to code!

🍋 Arrays

Arrays are mutable ordered structures of the same type. We will not go into the depths of using arrays, since most of the semantics are things you are familiar with from COMP1511. However, we will look at the basic usage of arrays.

```
1 // This is a array
2 const names = ['Hayden', 'Jake', 'Nick', 'Emily'];
3
4 console.log(`1 ${names}`);
5 console.log(`2 ${names[0]}`);
6 names[1] = 'Jakeo';
7 names.push('Rani');
8 console.log(`3 ${names}`);
```

1.4_arrays.js

```
1 1 Hayden,Jake,Nick,Emily
2 2 Hayden
3 3 Hayden,Jakeo,Nick,Emily,Rani
```

🍋 Arrays

We can use arrays with loops, too.

```
const items = ['a', 'b', 'c', 'd', 'e'];

let i = 0;
while (i < 5) {
  console.log(items[i]);
  i++;
}

for (let j = 0; j < 5; j++) {
  console.log(items[j]);
}

for (let k = 0; k < items.length; k++) {
  console.log(items[k]);
}
```

1.4_array_basic.js

🍋 Arrays

We can use arrays with loops, too.

```
1  function getEvens(nums) {
2    const evens = [];
3    for (let i = 0; i < nums.length; i++) {
4      if (nums[i] % 2 === 0) { // Why is this not == ??
5        evens.push(nums[i]);
6      }
7    }
8    return evens;
9  }
10
11 const allNumbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
12 console.log(getEvens(allNumbers));
```

1.4_functions.js

🍋 Arrays

Because Javascript is a higher level language, we have the ability to use a more concise and clear syntax when doing looping. You are not required to use this in the first few weeks of the course but we'd encourage all students to move toward this.

```
1  const items = ['a', 'b', 'c', 'd', 'e'];
2
3  // prints 0, 1, 2, 3, 4
4  for (const i in items) {
5    console.log(items[i]);
6  }
7
8  // prints a, b, c, d, e
9  for (const item of items) {
10   console.log(item);
11 }
12
13 console.log(items.includes('c'));
```

1.4_array_advanced.js

🍊 Objects

Objects are mutable associative structures that may consist of many different types. They are similar to C-style structs.

You can use them when you need a collection of items that are identified by a string description, rather than a numerical index (arrays).

```
1  const student = {
2    name: 'Emily',
3    score: 99,
4    rank: 1,
5  };
6
7  console.log(student);
8  console.log(student.name);
9  console.log(student.score);
10 console.log(student.rank);
11
12 student.height = 159;
13 console.log(student);
```

1.4_objects.js

🍊 Objects

We can create and populate objects different ways.

```
1 const userData = {};
2 userData.name = 'Sally';
3 userData.age = 18;
4 userData.height = '187cm';
5 console.log(userData);
```

1.4_object_basic1.js

```
1 const userData = {
2   name: 'Sally',
3   age: 18,
4   height: '187cm',
5 };
6 console.log(userData);
```

1.4_object_basic2.js

Both of these programs would print `{ name: 'Sally', age: 18, height: '187cm' }`

🍊 Objects

We can mix the two methods, and also use alternative syntax with assigning.

```
1 userData.prop = 1;
2 userData['prop'] = 1;
```

Or in a more full example.

```
1 // You can assign more keys even
2 //  after creation
3 const userData = {
4   name: 'Sally',
5   age: 18,
6 };
7 userData.height = '187cm';
8
9 console.log(userData);
```

1.4_object_more1.js

```
1 // You can reference keys with either
2 //  obj.key or obj['key']
3 const userData = {};
4 userData.name = 'Sally';
5 userData.age = 18;
6 userData.height = '187cm';
7 console.log(userData);
```

1.4_object_more2.js

🍊 Objects

We can also get various properties of an object using the `Object` functions.

```js
const userData = {
  name: 'Sally',
  age: 18,
  height: '187cm',
};

const keys = Object.keys(userData);
const entries = Object.entries(userData);
const values = Object.values(userData);

console.log(keys);
console.log(entries);
console.log(values);
```

1.4_object_props_1.js

```
[ 'name', 'age', 'height' ]
[ [ 'name', 'Sally' ], [ 'age', 18 ], [ 'height', '187cm' ] ]
[ 'Sally', 18, '187cm' ]
```

🍊 Objects

We can also loop through objects and check if certain keys are in them.

```js
const userData = {
  name: 'Sally',
  age: 18,
  height: '187cm',
};

for (const key in userData) {
  console.log(key);
}

if ('name' in userData) {
  console.log('Has name key');
}
```

1.4_object_props_2.js

```
name
age
height
Has name key
```

🧐 Further Discussion Of Objects

The following code exhibits behavior you're probably not used to:

```
1 const arr = [1, 2, 3];
2 console.log(arr.length);
3 console.log(arr.includes(3));
```

1.4_object_model.js

"arr" is an array, but it also seems to have:

- A property `length` that we never set?
- Some kind of function that is being called?

Let's look at why this is.

🕛 Recap: Various Types In C

4 Bytes, No Functions

```
1 int a;
```

8 Bytes, No Functions

```
1 struct point {
2   int x;
3   int y;
4 }
```

- Simple types in C were basic types that occupied limited memory.
- Structs were collections of primitive types wrapped into an "object".
- We would create instances of these "objects" and then access properties of them.
- We can expand this concept into Javascript.

🍅 Everything In Javascript Is An Object

Many Bytes, Some Functions

```
1 const arr = [1,2,3];
```

- In Javascript, basically every data type acts like an "object"
- This array is an object.
- An "object" being a data type that:
  - Contains 0 or more properties (/attributes)
  - Contains 0 or more functions (/methods)

To oversimplify: It's structs with functions

🍅 Everything In Javascript Is An Object

Can I Define My Own Objects?

- Yes! But we don't cover that in this course.

Does That Make Javascript Object-Oriented?

- C is a purely procedural language
- Java is a purely object-oriented language
- Javascript is a procedural language with OO capabilities

👯 Tying Some Things Together

Let's try some lists of objects.

```
const userData = [
  {
    name: 'Sally',
    age: 18,
    height: '186cm',
  }, {
    name: 'Bob',
    age: 17,
    height: '188cm',
  },
];

const keys = Object.keys(userData);
const entries = Object.entries(userData);
console.log(keys);
console.log(entries);
```

1.4_object_loop1.js

```
[ 'name', 'age', 'height' ]
[ [ 'name', 'Sally' ], [ 'age', 18 ], [ 'height', '187cm' ] ]
[ 'Sally', 18, '187cm' ]
```

👯 Tying Some Things Together

Let's try some lists of objects.

```
const userData = [
  {
    name: 'Sally',
    age: 18,
    height: '186cm',
  }, {
    name: 'Bob',
    age: 17,
    height: '188cm',
  },
];

for (let i = 0; i < userData.length; i++) {
  console.log(`${userData[i].name}'s properties are:`);
  console.log(`  name: ${userData[i].name}`);
  console.log(`  age: ${userData[i].age}`);
  console.log(`  height: ${userData[i].height}`);
}
```

1.4_object_loop2.js

👯 Tying Some Things Together

Let's try some lists of objects.

```
const userData = {
  Sally: {
    age: 18,
    height: '186cm',
  },
  Bob: {
    age: 17,
    height: '188cm',
  },
};

for (const key in userData) {
  console.log(`${key}'s properties are:`);
  for (const key2 in userData[key]) {
    console.log(`  ${key2}: ${userData[key][key2]}`);
  }
}
```

1.4_object_loop3.js

```
Sally's properties are:
age: 18
height: 186cm
Bob's properties are:
age: 17
height: 188cm
```

👯 Tying Some Things Together

Let's try more lists of objects.

```
1  const student1 = { name: 'Hayden', score: 50 };
2  const student2 = { name: 'Nick', score: 91 };
3  const student3 = { name: 'Emily', score: 99 };
4  const students = [student1, student2, student3];
5
6  console.log(students);
7
8  // Approach 1
9  const numStudents = students.length;
10 for (let i = 0; i < numStudents; i++) {
11   const student = students[i];
12   if (student.score >= 85) {
13     console.log(`${student.name} got an HD`);
14   }
15 }
16
17 // Approach 2
18 for (const student of students) {
19   if (student.score >= 85) {
20     console.log(`${student.name} got an HD`);
21   }
22 }
```

1.4_combining.js

🦻 Feedback

Or go to the form here.