JavaScript pt. 1: Syntax and Concepts

Kianoosh Abbasi

CSC309 Fall 2022

Some content is from Dr. Sadia Sharmin's slides of CSC309 Winter 2021: www.rainsharmin.com

Mahsa Amini

So far

- How web works
  Client/server – request/response - HTTP
- HTML
  Tags: headers, inputs, etc.
- CSS Styles
  Selectors, spacing, layout

This week

- Intro to JavaScript
- Objects and functions
- Scope
- Closures
- Arrow functions

Front-end so far

- HTML: Describes what should be on our page
- CSS: Describes how elements should look like
- But the web age is not interactive!
  We need something that responds to events and user actions
- JavaScript: a language that browsers understand!

Netscape - [Version 2.02]

File Edit View Go Bookmarks Options Directory Window Help

Back Forward Home Reload Images Open Print Find Stop

Location: about:

What's New! What's Cool! Handbook Net Search Net Directory Software

Netscape Navigator (TM)

Version 2.02

This software is subject to the license agreement set forth in the license. Please read and agree to all terms before using this software.

Report any problems through the feedback page.

NETSCAPE

JAVA COMPATIBLE

RSA PUBLIC KEY CRYPTOSYSTEM

This version supports International security with RSA Public Key Cryptograph

Any provision of Netscape Software to the U.S.Government is with "Restricted rights" as follows: Use, duplication or disclosure by the Government is subje

Java vs JavaScript

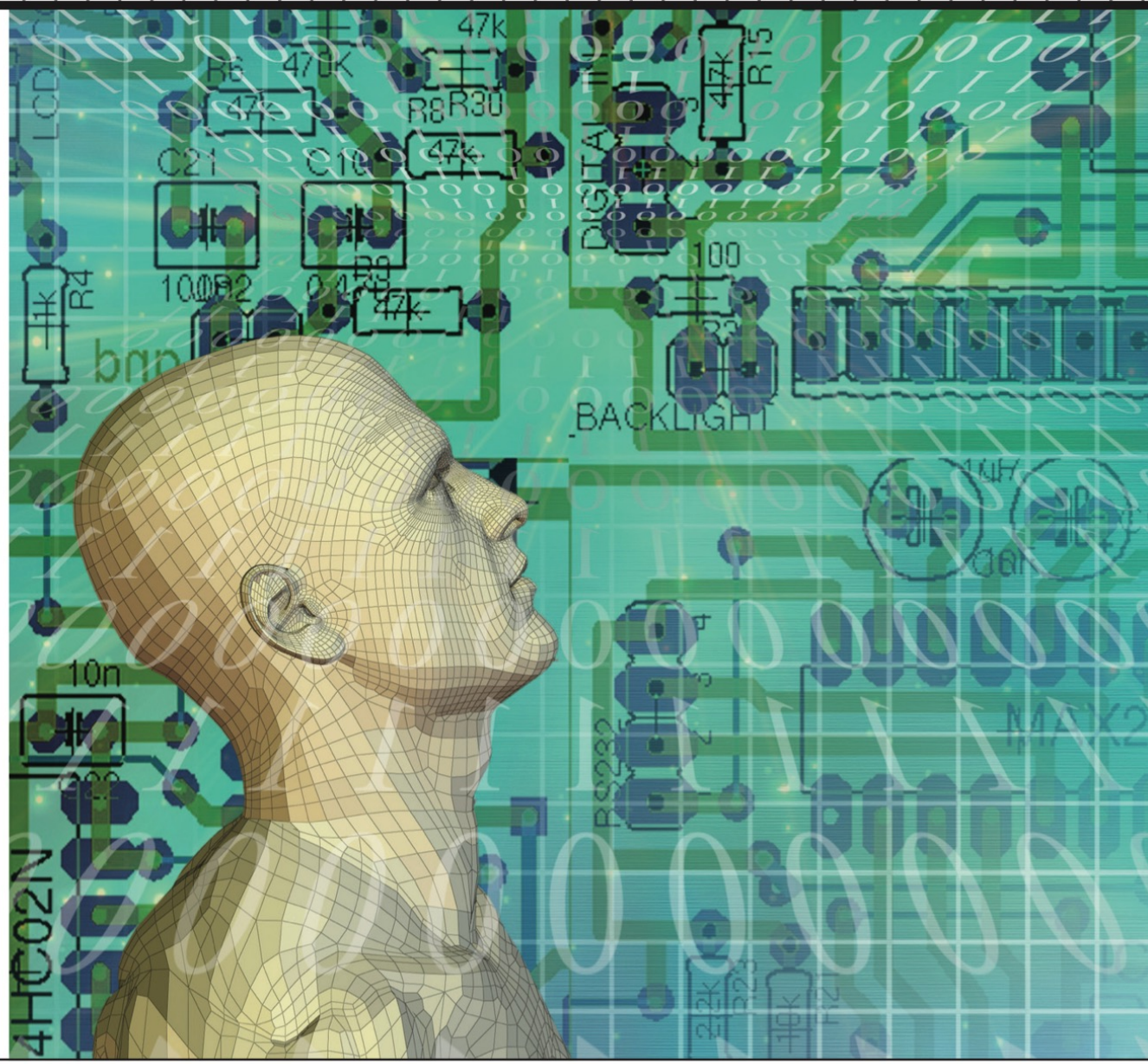
- Eich's script language had a somewhat similar syntax to Java

- Netscape-Sun deal:
  - Netscape browser will support Java apps
  - Eich's language will be called JavaScript!

- No further relevance between Java and JavaScript!

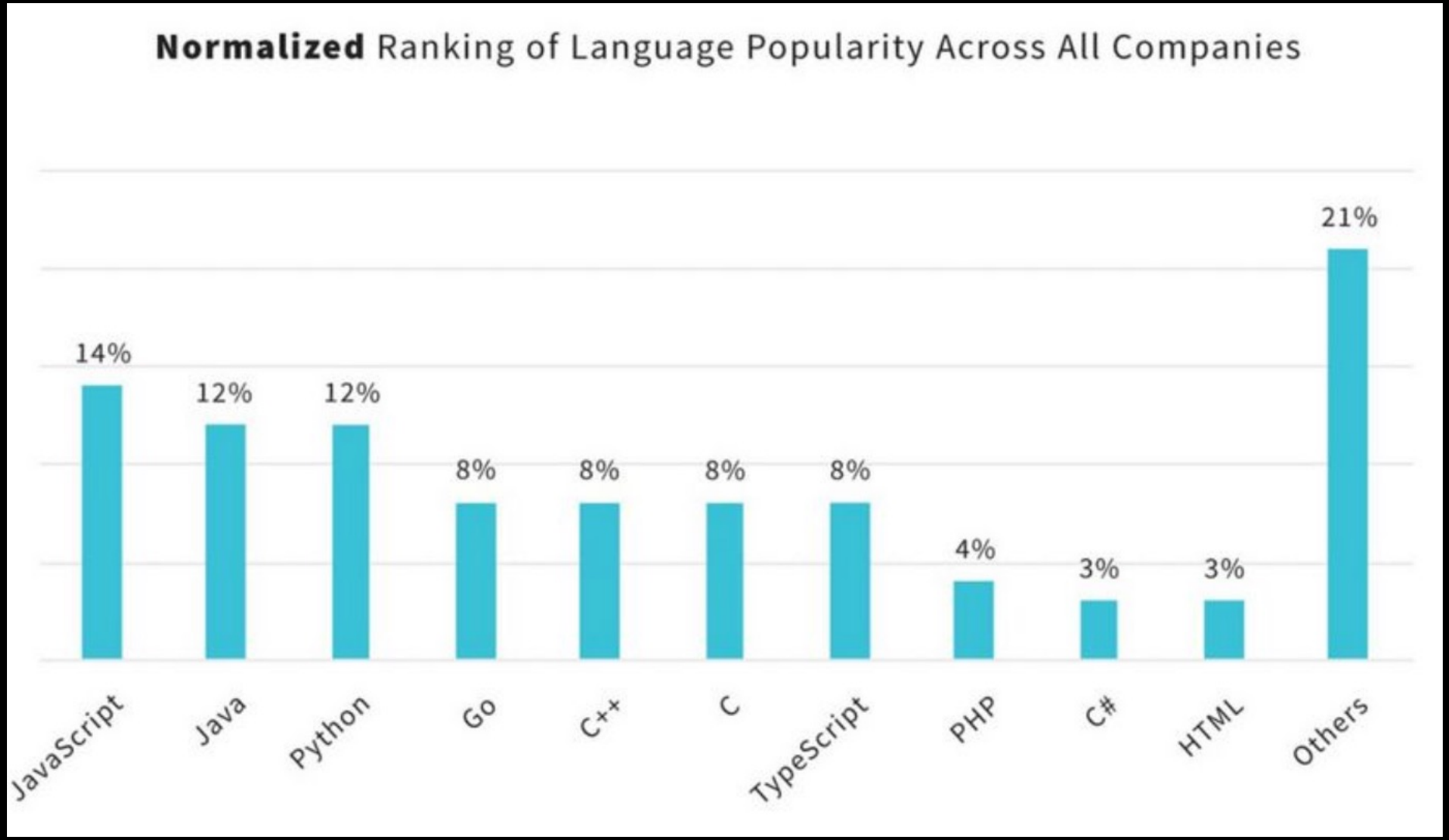
JavaScript: Designing a Language in 10 Days

Charles Severance
University of Michigan



There we go!

Visit: https://madnight.github.io/githut



Source: https://solutionshub.epam.com/blog/post/programming-language-popularity-on-github

- JavaScript is a scripting language
  - Interpreted at runtime
  - No JAR or exe file
- Almost all browsers have a JS interpreter
  - Run JS code accompanying the HTML (i.e., static files)
- Is it exclusive to client-side web applications?
- Answer: NO!
  - Example: NodeJS (more on that later)
  - Moreover, JS is just a language! You sort an array of integers with it

Syntax

- Declaring variables

```
var x = 5;
var y = "hello";
console.log(x + y);
```

- Data types:

  Number, string, boolean, undefined

  Object, function

- JS is dynamically-typed (like Python)

Objects

- Examples:

```
var cars = ["Saab", "Volvo", "BMW"];
var person = {firstName: "John", lastName: "Doe", age: 50,
eyeColor: "blue"};
var ref = null;
```

- Note: null is different from undefined

  `typeof(null)` returns object, while `typeof(undefined)` returns undefined!

Properties

- Examples:

```
person.firstName = "Joe";
person["lastName"] = "Jordan";
cars[0] = 1;
cars.push(2323);
```

- Objects (including arrays) are the only mutable types in JS

Functions

- Syntax:

```
function name(parameter1, parameter2, parameter3) {
  // code to be executed
}
```

- Properties can be functions (methods)

```
var obj = {f: function(x) {
   return x + 2
}}

cars.clear = function(){
    this.length = 0;
}
```

Classes

Visit https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Classes

- A template for creating objects
- Are in fact special functions

  Check `typeof(Rectangle)`
- Classes support inheritance

```
class Rectangle {
  constructor(height, width) {
    this.height = height;
    this.width = width;
  }
  // Getter
  get area() {
    return this.calcArea();
  }
  // Method
  calcArea() {
    return this.height * this.width;
  }
}

const square = new Rectangle(10, 10);

console.log(square.area); // 100
```

Conditions

- If statements:

```
if (typeof(cars[0]) === "number" && cars[0] < 0)
    cars[0] *= -1;
else
    console.log("Bad element");
```

- Important: notice `===`

  Visit https://codeahoy.com/javascript/2019/10/12/==-vs-===-in-javascript/

More statements

- While loops:

```
while (cars.length > 0){
    cars.pop();
}
```

- Switch statement:

```
switch(cars[0]){
    case 1:
        console.log("int");
        break;
    case "name":
        console.log("str");
        break;
    case x:
        console.log("var " + x);
        break;
    default:
        console.log("none");
}
```

For loops

- Classic for loop:

```
for (var i=0; i<10; i++)
    console.log(i * i * i);
```

- Iterable objects:

```
for (name of names)
    console.log("There is a " + name)
```

- Array-specific forEach:

```
names.forEach(function(index, name){
    console.log(name + " at index " + index);
})
```

Scope

Visit: https://www.w3schools.com/js/js_scope.asp

- Three types of scope:
  - Global scope
  - Function scope
  - Block scope

- Global scope
  - Outside any function
  - Variables can be accessed from anywhere in the program

Function scope

- Variables defined anywhere inside a function are local to that function
- Can be used anywhere inside that function
- Cannot be used outside that function

```
// code here can NOT use carName

function myFunction() {
  var carName = "Volvo";
  // code here CAN use carName
}

// code here can NOT use carName
```

Block scope

- To limit a variable to its block inside the function, use let

```
function f(n){
    if (n > 10){
        var tmp = 2;
    }
    // tmp CAN be accessed here
}
```

```
function f(n){
    if (n > 10){
        let tmp = 2;
    }
    // tmp can NOT be accessed here
}
```

Let vs var

- At global and function scopes, let and var work the same
- var supports redeclaration, while let does not
- Both support re-assignment. Use const to disallow it
- let is more like regular variables in other languages
  - Preferred over var

Let vs var

- let does not support redeclaration
- But what happens in a for loop?
  - It is in fact redeclared each time
- Not the case with var

Let vs var

- What is the difference between these two codes?

- The top code print the same value of i

```
for(var i = 1; i <= 5; i++) {
    setTimeout(function() {
        console.log('Value of i : ' + i);
    },100);
}


for(let i = 1; i <= 5; i++) {
    setTimeout(function() {
        console.log('Value of i : ' + i);
    },100);
}
```

Closures

Visit https://medium.com/@prashantramnyc/javascript-closures-simplified-d0d23fa06ba4

- What do variables X and Y store?
- What is the scope of variables a, b, c
- Local variables should be destroyed at the end of function

```
function outer() {

var b = 10;
var c = 100;

   function inner() {

         var a = 20;
         console.log("a= " + a + " b= " + b);


         a++;
         b++;


    }
   return inner;
}

var X = outer();  // outer() invoked the first time
var Y = outer();  // outer() invoked the second time
//end of outer() function executions
```

Closures

- inner captures variable b from outer

- Output:

```
a=20 b=10
a=20 b=11
a=20 b=12
a=20 b=10
```

```
function outer() {

var b = 10;
var c = 100;

   function inner() {

         var a = 20;
         console.log("a= " + a + " b= " + b);

         a++;
         b++;

    }
   return inner;
}

var X = outer();  // outer() invoked the first time
var Y = outer();  // outer() invoked the second time
//end of outer() function executions

X(); // X() invoked the first time
X(); // X() invoked the second time
X(); // X() invoked the third time

Y(); // Y() invoked the first time
```

Arrow functions

- A more convenient way to define functions
- Almost equivalent to regular functions
  - More on that later

```
function regular(a, b){
  return a + b;
}

const arrow = (a, b) => {
    return a + b;
}

const conciseArrow = (a, b) => a + b;
```

Simplify even further

- Today, for loops and if statements are rarely used
- Instead of a for loop, use forEach or map
- Example:

```
var names = ["ali", "hassan"]
names.forEach((item, index) => console.log(item + " at " + index))
upper = names.map(item => item.toUpperCase())
```

Simplify even further

- Take out elements with a specific condition
- Use the filter method instead of for loop and if
- Example:

```
let students = [{name: "John", id: 1}, {name: "Ali", id:2}]
let john = students.filter(item => item.name === "John")
```

Simplify even more further!

- reduce lets you do a lot of cool things with just 1 inline arrow function

- Example:

```
let maxCredit = employee.reduce((acc, cur) =>
Math.max(cur.credit, acc), Number.NEGATIVE_INFINITY)
```

Power of arrow functions!

Regular functions

```
var totalJediScore = personnel
  .filter(function (person) {
    return person.isForceUser;
  })
  .map(function (jedi) {
    return jedi.pilotingScore + jedi.shootingScore;
  })
  .reduce(function (acc, score) {
    return acc + score;
  }, 0);
```

Arrow functions

```
const totalJediScore = personnel
  .filter(person => person.isForceUser)
  .map(jedi => jedi.pilotingScore + jedi.shootingScore)
  .reduce((acc, score) => acc + score, 0);
```

Source: https://medium.com/poka-techblog/simplify-your-javascript-use-map-reduce-and-filter-bd02c593cc2d

Subtlety

- Regular functions have their own this value
- The object that called the function
  - Methods and event listeners: the actual object/element
  - Global function: global object (window)
- Arrow functions do not have their own this

- Do not use arrow functions as event listeners or object methods

  You can use them as class methods though. PERFECTLY WEIRD ISN'T IT?

- However, unlike regular functions, they can bind (capture) this like any other closure value

```
const person = {
    name: 'Kianoosh',
    sayHi() {
        setTimeout(() =>
            console.log(this.name + " says hi!"), 500);
    }
}
```

- For more information, visit https://www.javascripttutorial.net/es6/when-you-should-not-use-arrow-functions/

Destructuring

Visit https://dmitripavlutin.com/javascript-object-destructuring/

```
const hero = {
  name: 'Batman',
  realName: 'Bruce Wayne'
};

const { name, realName } = hero;

name;     // => 'Batman',
realName; // => 'Bruce Wayne'
```

```
const heroes = [
  { name: 'Batman' },
  { name: 'Joker' }
];

const names = heroes.map(
  function({ name }) {
    return name;
  }
);

names; // => ['Batman', 'Joker']
```

```
const hero = {
  name: 'Batman',
  realName: 'Bruce Wayne'
};

const { name, ...realHero } = hero;

realHero; // => { realName: 'Bruce Wayne' }
```

Sessions

- If your project uses session auth, browser already stores and sends the cookies headers

- If it uses token auth, you are responsible for storing and using the token

```
localStorage.setItem('access_token', access_token);
localStorage.getItem('access_token');
```

- Set Authorization header with the appropriate value

This week

- Intro to JavaScript
- Objects and functions
- Scope
- Closures
- Arrow functions

Next week

- DOM
  Getting and manipulating elements
- jQuery
- Asynchronous requests: Ajax
- Event loop
- Fetch API and Promises