ACCESS LABS

POWERED BY // FLATIRON SCHOOL

Software Engineering Immersive

Syllabus & Program Guide

Last Updated: 2/20/2018

Table of Contents

Overview

Access Lab's Initiative - Powered by Flatiron School

Being a software engineer requires more than knowing how to code or build a web app. Over 15 challenging weeks, students learn to think, and build, like software engineers—from developing coding proficiency to gaining an understanding of how products are designed and managed. In each three-week curriculum module, students develop key skills through interactive labs, lectures, and close collaboration, showcasing progress through Portfolio Projects. While the bulk of the material covered encompasses the Ruby and JavaScript ecosystems, we carefully designed our curriculum to prepare students to launch software engineering careers, independent of any specific language or technology.

By undertaking this rigorous but rewarding program, students:

- develop a foundation in programming fundamentals
- conquer the concepts of object-oriented programming
- work with APIs (Application Programming Interfaces)
- become proficient in database modeling and ORM (Object Relational Mapping)
- understand MVC (Model-View-Controller), a pattern used by frameworks like Rails to build large-scale applications
- execute application deployment
- learn how code fits into a larger product from NYC's top entrepreneurs
- study Computer Science fundamentals for increased technical depth

By the completion of the program, students have done much more than simply build technical skills: they have maintained technical blogs to show they can credibly talk tech; they have become a part of the NYC tech community; they have amassed an impressive portfolio of unique, functional web applications to show employers as they enter the job-search phase.

Proven Results

Access Labs is powered by Flatiron School who has helped over 1,000 students get jobs as software engineers. Since 2012, FLatiron School has backed up their 97% employment rate with independently-verified Outcomes Reports.

The Power of Learning Ruby and JavaScript

Why do we teach these languages to aspiring software engineers?

Readability

Programming is about abstractions and expressions: the mechanics of code are universal and exist in all modern languages. Much of the initial difficulty in learning programming stems from the learning curve necessary to gain comfort with a language's syntax.

Ruby was specifically designed by its inventor Yukihiro Matsumoto to make programmers happy and it's delivered upon that objective: Ruby is expressive, accessible, and reads remarkably like English, allowing new programmers to focus immediately on the fundamental concepts and logic, rather than basic syntax. As such, even beginners can start building right away. We find Ruby to be extremely effective at helping students learn how to think like programmers, break problems down, express themselves technically, abstract ideas, and work together with other programmers.

It's also easy to get started with JavaScript. Programs written in C++ and Java must be compiled to machine code before running. In contrast, JavaScript programs compile at runtime—you write and it runs—making it simple to start building.

Career Flexibility

Learning to code and evolving as a programmer is a lifelong endeavor. No matter what language you learn first, you will have to learn other languages throughout your career in order to keep pace with changing technology. Starting off your coding education by learning both Ruby and JavaScript not only makes you a more versatile—and employable—programmer immediately; it also prepares you for the essential task of continuing to learn throughout your career as new languages, frameworks, and technologies are released.

Open Source

Both Ruby and JavaScript have nurtured vibrant, supportive open source communities—there are over 900 Ruby groups on Meetup.com, totaling over 500,000 members worldwide; JavaScript has over 4,000 groups on Meetup.com with over two million members. Ruby also has a huge and useful ecosystem of over 60,000 libraries. This will allow you to leverage free, publicly-available tools to build applications with complexity and real-world use beyond what you could approach otherwise.

JavaScript is Essential for the Modern Web

JavaScript is the language that fundamentally changed the web from static, text-filled pages into the interactive experience it is today—and, as one of the most popular languages out there, it's still bringing websites to life in new, exciting ways. As a front-end developer, you simply can't avoid it. And with Node.js allowing JavaScript to run client- and server-side, it's a valuable tool for full-stack developers as well.

Curriculum Overview

Getting Started

Before starting the Access Labs Initiative program, students must complete an initial technical assessment focused on their choice of JavaScript or Ruby—this will require some prior programming experience. In order to prepare, we recommend beginning to work on Flatiron School's online Bootcamp Prep course, accessible via FlatironSchool.com. If accepted into the program, students must complete a 100-hour Introductory Programming course before the first day of class. This pre-work ensures students come in prepared and are able to keep pace with the class.

Mastery-based progression

The Access Labs program is broken into five three-week curriculum modules. Each module concludes with a comprehensive project meant to bring together students' learnings and demonstrate them in their portfolios. Students work in teams and directly with instructors to ensure they've mastered key concepts before progressing. If students need additional support, they receive additional direct mentorship, and, if necessary, they'll have the opportunity to repeat a module at no extra cost.

Lifelong Learning

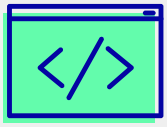
Graduates will join a growing network of successful software engineers. Engaging with our community doesn't stop at graduation—and nor do opportunities to learn. Since the program is powered by Flatiron School, alumni will gain lifetime access to additional curricula on Flatiron School's online platform, Learn.co, including iOS Immersive with Swift and Objective-C, Computer Science in Java, and Advanced JS with Angular 1. Additionally, students enjoy lifetime access to Flatiron School's Career Services team and extensive employer network as they approach future career changes.

CURRICULUM OVERVIEW

Full Stack Web Development

Flatiron School's Full Stack Web Development Curriculum was designed to give students the necessary expertise in both back-end and front-end programming technologies to become full-stack developers. It's a more extensive course of study than the average school offers – but then we expect more of our students.

Pre-work	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	Post-grad



Module 1: Programming Fundamentals (weeks 1-3)

After diving into HTML5 & CSS, students get comfortable with object-oriented programming, learning to read websites with Ruby and save data to a database with SQL and Object Relational Mappers.

Module 2: Web Frameworks (weeks 4-6)

Students learn two key Ruby frameworks, first mastering the fundamentals of web programming with Sinatra before experiencing how quickly they can build incredible apps with Rails.

Module 3: JavaScript (weeks 7-9)

Students gain a thorough understanding of Javascript and functional programming – crucial for front-end devs – and start to build their own version of React before moving onto the framework itself.

Module 4: Front-end Frameworks (weeks 10-12)

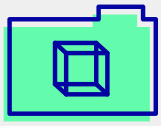
Students learn to build productive, scalable front-ends with React and Redux, creating slick, functional, reactive code with Redux as a state manager and Rails as the back-end JSON API.

CURRICULUM OVERVIEW

Solo Projects

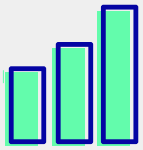
After completing four curriculum modules focused on group projects, students work with instructors to come up with solo project concepts and spend three full weeks building truly sophisticated applications on their own. Students receive plenty of instructor feedback along the way while diving deep into various advanced technologies needed to bring their concepts to life. Languages and technology used vary by project and are limited only by the students' motivation. These are example technologies students have used previously.

Pre-work	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	Post-grad



Databases

Students go deep into NoSQL databases like Mongo and gain a better understanding of working with databases at scale.



Performance

Students focus on design patterns and use performance-monitoring tools to take a data-driven approach to increasing application performance.

Front-end Frameworks

Students take a deeper dive into React and develop advanced HTML & CSS skills, including SASS and D3.

Mobile Apps with React Native

Our world is mobile and our platform is the Web. Students have the ability to build on previous concepts to take their portfolios to the next level.

CURRICULUM OVERVIEW

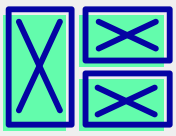
Product Series

No piece of software is build by one person. Product Series curriculum gives students a chance to look beyond the command line and see how code fits into a larger product or organization. Throughout this program, students hear from some of the best entrepreneurs and builders in NYC to learn what it takes to go from prototype to product adoption.

Pre-work	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	Post-grad

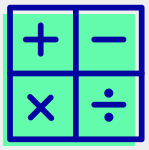
Finding Product-Market Fit

Building a great product requires understanding its users. Students learn about conducting user research, defining a value proposition, and leveraging data to drive user behavior.



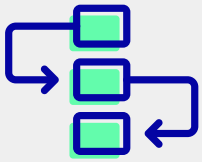
UX & Design

Students delve into User Experience, learning how to bridge physical and digital experiences and develop data-driven, habit-forming UX design strategies.



Business Fundamentals

Students explore the business factors that surround product, learning about business models, KPI tracking, and product management for startups and enterprises.



Workflows

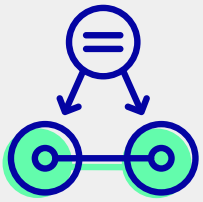
Our world is mobile and our platform is the Web. React Native allows students to build complex native mobile apps using web-based technologies.

CURRICULUM OVERVIEW

Computer Science

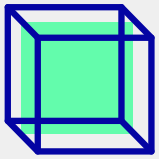
Building complex applications at web-scale requires an understanding of algorithms and Computer Science fundamentals. Studying this discipline gives students an advantage in technical interviews and lays a powerful foundation for students to increase their technical depth throughout their careers.

Pre-work	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	Post-grad



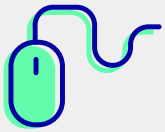
Algorithms

To understand the resource constraints that affect software performance, students study List Searching and Sorting, Graph Manipulation, and Algorithmic Thinking and Algorithmic Analysis.



Data Structures

To understand the ramifications of how data is stored and accessed, students build their knowledge of Hash Tables, Stacks, Queues, Trees, and Graphs.



Technical Interview Training

Through extensive practice problems and mock interviews with instructors, students become prepared for the CS component of technical interviews.

The Full Stack Journey

Access Labs is powered by Flatiron School's Full Stack Web Development Curriculum Below, we take a closer look at the exact technologies we cover in this curriculum.

Ruby

Students begin their programming journey at Access Labs with the expressive, easy-to-read Ruby language. Initial topics build in complexity and provide the foundation for the rest of the course. In this unit, students:

- Explore version control using git commands, including: cloning, branching, merging, rolling back commits, forking, and submitting pull requests.
- Learn fundamental concepts in programming, including repls, methods, loops, variables, variable scope, conditionals, blocks and iterators, case statements, arrays, scope, hashes, regular expressions, iterators, enumerables, data structures, nesting, and more.
- Learn to embrace error messages as clues and gain a fundamental appreciation for failure as the only way to learn and progress.
- Gain experience in debugging with various gems and tools designed to track down issues in code.

HTML & CSS

By exploring HTML & CSS, students master the basic building blocks of how the web is rendered and become fluent in the language that makes the web beautiful. In this unit, students:

- Learn how to conceive of and build UIs for web apps by writing well-structured HTML and CSS.
- Use Sass to create efficient and organized front-ends

Object Orientation

In Object-Oriented Programming, programmers define not just a data structure's data type, but the functions that can be applied to it as well. In this unit, students:

- Gain experience with Object-Oriented Programming (OOP).
- Understand how OOP allows programmers to bundle code and create reusable objects and methods, which allows for increasing complexity in software.

Object Relational Mapping & SQL

ORM (Object Relational Mapping) allows programmers to query and manipulate data from a database using an object-paradigm. In this unit, students:

- Learn to write and manipulate data using the Ruby language.
- Gain an appreciation for the structure of a database, how to map out tables, and the difference between the various table relationships.
- Learn how to wireframe database structures, as well as how to link their applications to a database.
- Cover SQL, domain modeling, relational database theory, schema architecture, and the Object Relational Model, including the ActiveRecord pattern.

Rack

This unit is designed to give students an understanding of HTTP and how the Internet works, as implemented through the Ruby web interface of Rack. In this unit, students:

- Build their own HTTP servers and learn how the request / response model of the web works.
- Create servers that listen to HTTP requests and respond with well-formed HTML responses.
- Learn to understand the web with the few abstractions provided by the tool set.

Sinatra

Sinatra is a Domain Specific Language (DSL) written in Ruby for building web applications on top of Rack. In this unit, students:.

- Gain exposure to design patterns in web applications.
- Cover topics including architectural patterns such as REST (Representational State Transfer), MVC (Model-View-Controller), HTML Forms, ERB (Embedded Ruby) as well as template rendering and application environments.

Rails

With a foundation in the Ruby language as well as the architecture of the World Wide Web, students utilize Rails to build complex, functional web applications from the ground up. In this unit, students:

- Learn the file structure of Rails, how to set up their own databases, how to draw routes and create Rails forms, gain an understanding of the asset pipeline, and bring it together by integrating front-end design skills.
- Take on more advanced concepts such as authorization, validation, and callbacks.
- Spend time building out their own Rails applications, moving through the entire process from ideation to execution.

JavaScript

JavaScript powers the user experience of the web. In this unit, students:

- Learn the basics of JavaScript syntax, its functional architecture, and different approaches to the object model.
- Learn the Document Object Model (DOM) Javascript API provided by the browser to dynamically interact with HTML.
- Focus on jQuery, the most popular JavaScript library, to learn how to collect user input, manipulate the DOM with animations and injection, and send Asynchronous AJAX requests for a rich user experience.
- Explore popular JavaScript frameworks including AngularJS, Ember, and more.

React

Using plain JavaScript with large web applications quickly becomes unruly and unmanageable. Initially created by Facebook, React is the premier JavaScript framework for building fast web user interfaces. In this unit, students:

- Learn the fundamentals of components.
- Learn how to build the basics of React before they are taught how to use React itself.
- Build a minimal React and conquer the complexities of React before quickly moving into learning about state management with Redux.
- Build applications that effortlessly consume APIs, render data quickly, and scale as application complexity increases.

Node and Express

Built on top of the V8 Javascript Runtime, the Node Javascript ecosystem is becoming a popular and useful tool for asynchronous and real-time application development. In this unit, students:

- Focus on building real-time web application servers with Node and the Express.js framework.
- Learn how to build full stack JavaScript web applications from end-to-end with a focus on WebSockets and the real-time web.

What Makes a Good Engineer?

While the linear progression of our curriculum is focused on building technical skills, our aim is to teach students how to become software engineers—which is distinct from simply knowing how to code. Students engage in a number of activities that hone their communication and collaboration skills and immerse them in the technical community, helping them build the foundation needed to grow as software engineers in the future.

Portfolio Projects

At the conclusion of each program module, students build advanced Portfolio Projects to demonstrate both the technical skills they've gained in the module and their creativity. Previous projects have won prestigious tech awards, become MVPs for startups, and been presented at the White House. Portfolio Projects represent an opportunity for students to explore specific technologies that interest them while building a portfolio of fully functional web applications to impress employers.

Active Github Profile

GitHub is the modern software engineer's resume. Students push every line of code they write during the Access Labs program to GitHub through Flatiron School's proprietary platform, Learn.co, giving them an extensive profile to show employers and fellow engineers.

Robust Blog

All Access Labs students maintain technical blogs to show they can not only write code, but communicate how that code works – an essential skill for software engineers.

Technical Presentation

Students build their credibility as engineers and immerse themselves in NYC's technical community by attending – and challenging themselves to present – at technical Meetups and conferences.

How Do We Learn?

Access Labs students will learn using Flatiron School's Learn.co, the world's most sophisticated platform for teaching and learning code.

Use Real Tools

You can't learn real skills without real tools. We don't believe in contrived environments or multiple choice quizzes. Learn.co users set up a real development environment with our fast setup process and use a professional command line and Git-based workflow. You'll truly learn by doing.

Open Curriculum

The industry-tested curriculum for Access Labs has already given over 1,000 Flatiron School graduates the skills to become software engineers and thrive in their careers. And because the curriculum is open-source, it stays more current than any other. Students are encouraged to suggest changes directly from our online platform, and receive public credit for doing so. We continually improve our coursework in reaction to feedback and real-world changes, and our edits are supplemented by hundreds of student submissions each month.

Test-Driven Learning

Access Labs students learn usiing Test-Driven Development (TDD), where code requirements are defined before a program is written. Students complete lessons by writing code that meets requirements established by our curriculum. Tests are automated and descriptive, so students can learn by solving real problems and understanding not only when code is broken, but why.

Contact Us

For more information, please check out our website at accesslabs.org or contact us at admissions@flatironschool.com.