UDACITY

NANODEGREE PROGRAM SYLLABUS

Full Stack Web Developer

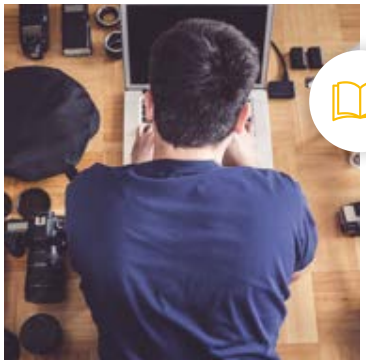
Overview

The goal of the Full Stack Web Developer Nanodegree program is to equip learners with the unique skills they need to build database-backed APIs and web applications. A graduate of this program will be able to:

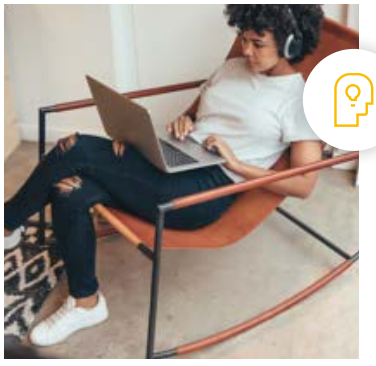
- Design and build a database for a software application
- Create and deploy a database-backed web API (Application Programming Interface)
- Secure and manage user authentication and access control for an application backend
- Deploy a Flask-based web application to the cloud using Docker and Kubernetes

This program includes 4 courses and 5 projects. Each project you build will be an opportunity to apply what you've learned in the lessons and demonstrate to potential employers that you have practical full-stack development skills.

Estimated Time: 4 Months at 5-10hrs/week



Prerequisites: Prior experience with Python, CSS/ HTML, and Git



Flexible Learning: Self-paced, so you can learn on the schedule that works best for you

Need Help? udacity.com/advisor Discuss this program with an enrollment advisor.

Course 1: SQL and Data Modeling for the Web

Master relational databases with the power of SQL, and leverage Python to incorporate database logic into your programs.

Course Project
Design a Venue Booking Database

For your first project, you'll be building out the data models and database for an artist/venue booking application. The fictitious startup Fy-yur is building a website that facilitates bookings between artists who can play at venues, and venues who want to book artists.

This site:

- Lets venue managers and artists sign up, fill out their information, and list their availability for shows.
- Lets artists browse venues where they can play, and see what past/upcoming artists have been booked at a venue.
- Lets a venue manager browse artists that would like to play in their city, and see what past/upcoming venues where the artist has played/will be playing.

The goal of this project is to build out the data models for this booking application. A prototype design of the web app will be provided. You'll use SQLAlchemy and Postgresql to build out the data models upon which this site will rely. You'll write out both the raw SQL and SQLAlchemy commands to run for powering the backend functionality of the website.

	LEARNING OUTCOMES	
LESSON ONE	**Connecting and Interacting with Databases**	• Describe and explain the client-server model • Describe and explain the TCP/IP communication protocol • Describe and explain the base unit of database work: transactions • Install the PostgreSQL database management system • Create and manage Postgres databases with the psql client • Install the psycopg2 Python+Postgres database driver • Create and manage Postgres databases using the psycopg2 Python database driver

LESSON TWO	**Intro to SQLAlchemy and SQLAlchemy ORM Basics**	• Describe and explain the use cases for an Object Relational Mapping (ORM) library • Describe and explain the abstraction layers of SQLAlchemy • Connect to and manage a database using composable SQL expressions • Define data model objects with Python using SQLAlchemy ORM • Connect data models to a lightweight Flask web application • Build data models using different types of data
LESSON THREE	**SQLAlchemy ORM in Depth**	• Explore and retrieve data using the SQLAlchemy Model.query object • Create database sessions for executing database transactions • Execute database transactions within a connection session • Describe and explain the SQLAlchemy object lifecycle • Build a lightweight data app using SQLAlchemy • Describe and explain the Model-View-Controller (MVC) application architecture • Retrieve from data from a webform using Flask • Update data models using data migrations • Migrate data using Flask-Migrate and Flask-Script • Define and code relationships between tables and objects using SQLAlchemy • Implement database methods to query relationships between data models
LESSON FOUR	**Build a CRUD App with SQLAlchemy ORM - Part 1**	• Use the CRUD (Create, Read, Update, Delete) model to build a small database backed app • Capture user input from a webform to add and modify data to a database • Manage data using database sessions in an application controller
LESSON FIVE	**Migrations**	• Modify a data schema using Flask-Migrate and Alembic • Write migration scripts to update data schemas using Flask-Script
LESSON SIX	**Build a CRUD App with SQLAlchemy ORM - Part 2**	• Update database models using webforms and application routing • Delete information from a database using SQLAlchemy • Model and control relationships between different types of data objects • Implement one-to-many and many-to-many relationships using SQLAlchemy • Execute complex database queries on related data models

Course 2: API Development and Documentation

Learn how to use APIs to control and manage web applications, including best practices for API testing and documentation.

Course Project:
Trivia API

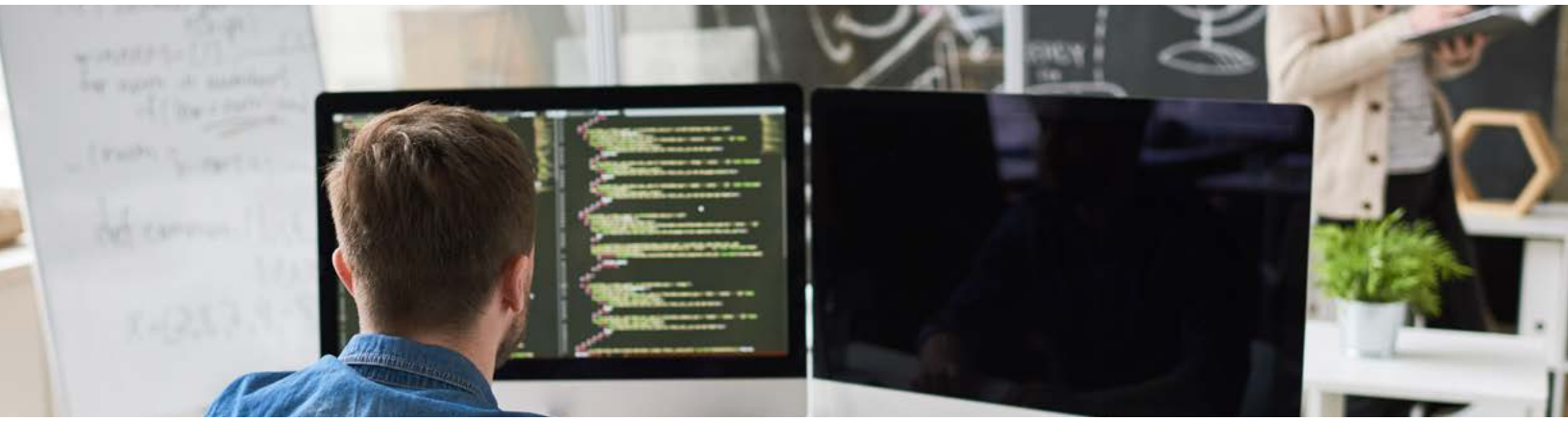
In this project, you will use the skills you've developed to build a Trivia API. The API will allow users to:

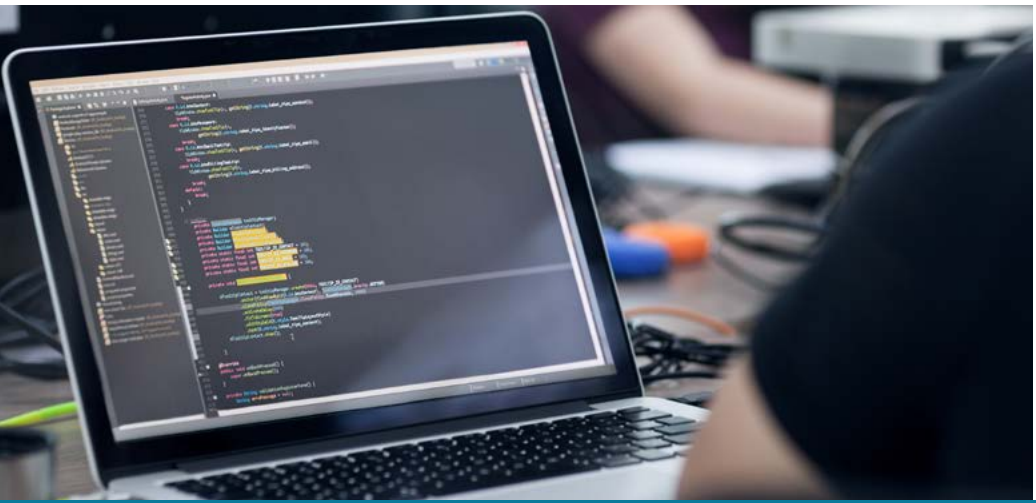
- Search for trivia questions and answers via category and difficulty
- Add new questions
- Modify the difficulty rating of questions.

The goal of this project is to use APIs to control and manage a web application using existing data models. You'll be given a set of data models and the application front end. Your task will be to implement the API in Flask to make the Trivia game functional.

	LEARNING OUTCOMES	
LESSON ONE	**Introduction to APIs**	• Describe and explain the definition and use cases of APIs (Application Programming Interface) • Describe and explain how APIs are used to connect application front ends to server backends
LESSON TWO	**HTTP and Flask Basics**	• Describe and explain the Hypertext Transfer Protocol (HTTP) • Describe and explain the components of an HTTP request • Describe and explain the different HTTP methods (verbs) • Describe and explain HTTP status codes • Request information from a server using cURL and HTTP requests • Install the Python Flask micro application framework • Set up and Configure a Flask application • Create a Flask endpoint (route)

LESSON THREE	**Endpoints and Payloads**	• Structure and Organize API Endpoints • Describe and explain Cross-Origin Resource Sharing (CORS) • Manage CORS requests using HTTP headers • Manage CORS controls using Flask-CORS • Parse request path and body from an HTTP request • Implement HTTP POST, PATCH and DELETE methods using Flask • Handle application errors using Flask
LESSON FOUR	**API Testing**	• Describe and explain the purpose and benefits of API testing • Test a REST API using Flask and unittest • Develop an application iteratively and safely using Test Driven Development (TDD)
LESSON FIVE	**API Documentation**	• Read and explore API documentation from a number of API developers • Write effective documentation for your own API





Course 3: Identity Access Management

Implement authentication and authorization in Flask and understand how to design against key security principle. You will also gain experience with role-based control design patterns, securing a REST API, and applying software system risk and compliance principles.

Course Project
Identity Access Management

In the third project of the program, you will build the backend for a coffee shop application. You'll add user accounts and authentication to your application and use role-based access management strategies to control different types of user behavior in the app. The application must:

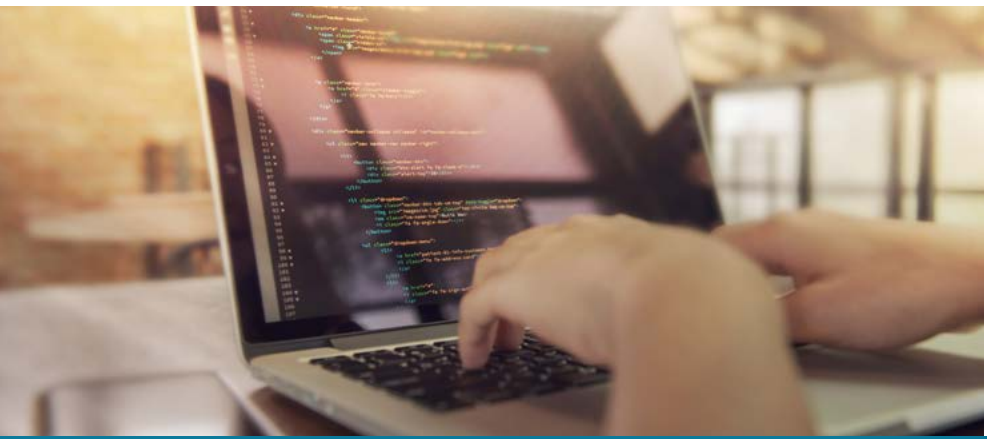
- Display graphics representing the ratio of ingredients in each drink.
- Allow public users to view drink names and graphics.
- Allow the shop baristas to see the recipe information.
- Allow the shop managers to create new drinks and edit existing drinks.

This project will give you a hands-on chance to practice and demonstrate your new skills, such as:

- Implementing authentication and authorization in Flask
- Designing against key security principles
- Implementing role-based control design patterns
- Securing a REST API
- Applying software system risk and compliance principles

		LEARNING OUTCOMES
LESSON ONE	**Foundations**	• Describe and explain the use cases and differences between authorization and authentication • Describe the problem of security and the risks of unsecured or improperly secured application systems • Describe different types of security attack • Inspect requests and responses for an application using Postman
LESSON TWO	**Authentication**	• Describe common methods for application authentication • Explain why passwords are not the ideal method for authentication • Implement an application authentication layer with Auth0 • Secure API communications using JSON Web Tokens (JWT)

	LEARNING OUTCOMES	
LESSON ONE	**Passwords**	• Describe the risks associated with password controlled systems • Mitigate access risks associated with SQL injection by validating and sanitizing database inputs • Secure database data in a database using standard encryption practices • Describe how an attacker can use rainbow tables to gain access to a system • Improve security of hashed passwords and encrypted data using the 'salt' method • Increase application security by using best practices to avoid logging and serializing sensitive data
LESSON TWO	**Authorization**	• Describe the concept of authorization and access control • Define 'permissions' in the context of an application • Constrain permissions in an application by using role-based access control (RBAC) • Define permission roles using Auth0 • Identify user permissions and roles from JWTs (JavaScript Web Tokens)
LESSON THREE	**Thinking Adversarially**	• Prevent accidental access to privileged information in Git repositories by using environment variables • Mitigate risks to Git master branch changes by developing in feature branches • Employ code review as a practice to mitigate security risks • Test API and authentication practices with integration testing • Describe common types of adversarial attacks on network systems.



Course 4: Server Deployment and Containerization

Develop an understanding of containerized environments, use Docker to share and store containers, and deploy a Docker container to a Kubernetes cluster using AWS

Course Project
Deploy a Flask App to Kubernetes Using EKS

In this project, you will create a container for your Flask web app using Docker and deploy the container to a Kubernetes cluster using Amazon EKS. By the end of the project, you will have deployed your application live to the world, where it should be accessible by IP address. You'll use automated testing to prevent bad code from being deployed and monitor your app's performance using AWS tools.

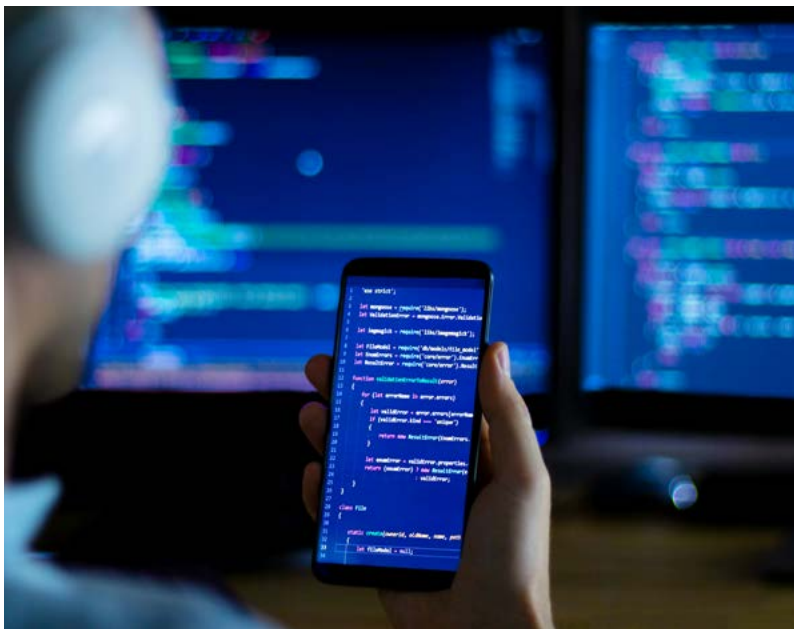
Course Project
Full Stack Web Developer Nanodegree Program Capstone

In this final capstone project, you will combine all of the new skills you've learned and developed in this course to construct a database-backed web API with user access control. You will choose what app to build and then you'll design and build out all of the API endpoints needed for the application and properly secure them for use in any front end application (web or mobile).

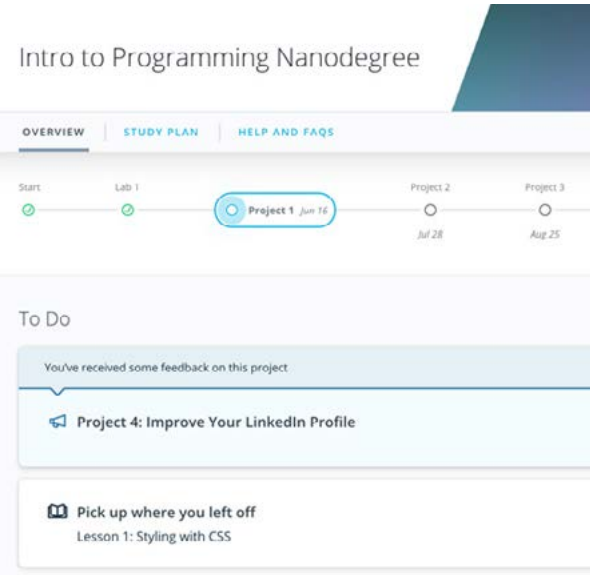
	LEARNING OUTCOMES	
LESSON ONE	**Containers**	• Describe and explain the benefits and use cases for containerized environments • Install Docker on a local machine • Define a container environment using a Dockerfile • Download and launch a Docker container • Store and share a docker container

LESSON TWO **Deployment**

- Describe and explain container orchestration, how it works and the general use case
- Describe and explain how Kubernetes manages container clusters
- Deploy a Docker container to a Kubernetes cluster using AWS EKS and the AWS command line interface (CLI)
- Manage Kubernetes clusters using the AWS CLI
- Implement Continuous Delivery (CD) and Continuous Integration (CI) with AWS CodePipeline and AWS CodeBuild



Our Classroom Experience

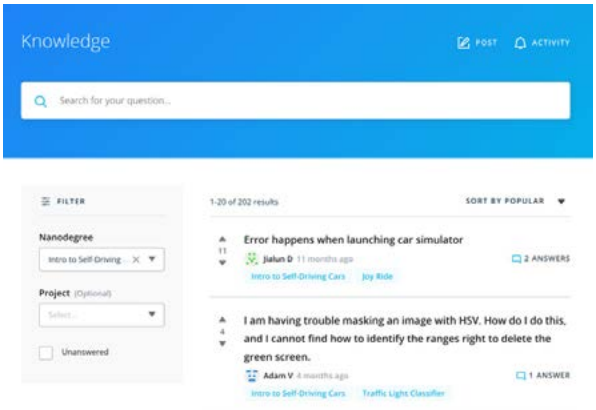


REAL-WORLD PROJECTS

Build your skills through industry-relevant projects. Get personalized feedback from our network of 900+ project reviewers. Our simple interface makes it easy to submit your projects as often as you need and receive unlimited feedback on your work.

KNOWLEDGE

Find answers to your questions with Knowledge, our proprietary wiki. Search questions asked by other students, connect with technical mentors, and discover in real-time how to solve the challenges that you encounter.

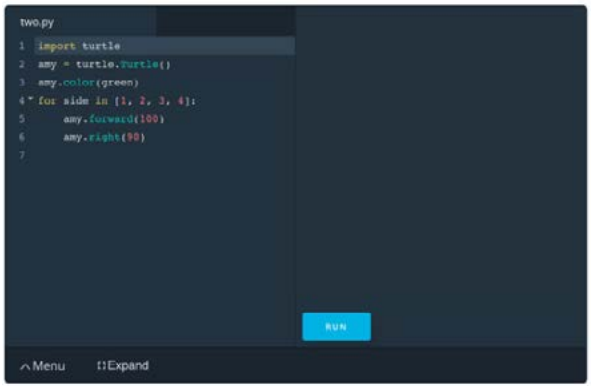


WORKSPACES

See your code in action. Check the output and quality of your code by running them on workspaces that are a part of our classroom.

QUIZZES

Check your understanding of concepts learned in the program by answering simple and auto-graded quizzes. Easily go back to the lessons to brush up on concepts anytime you get an answer wrong.



CUSTOM STUDY PLANS

Create a custom study plan to suit your personal needs and use this plan to keep track of your progress toward your goal.

PROGRESS TRACKER

Stay on track to complete your Nanodegree program with useful milestone reminders.

Learn with the Best

Amy Hua

INSTRUCTOR

Amy has 6+ years of experience as a software professional, building everything from data visualizations to self-driving cars. She's been a bootcamp instructor, StartupBus mentor, and Girls Who Code instructor.

Caryn McCarthy

INSTRUCTOR

Caryn has worked as a software developer and as Coach and Experience Manager at Code Next at Google. She is passionate about diversity and equity in tech, is always working to create positive impact in the tech industry and the world.

Gabriel Ruttner

INSTRUCTOR

Gabe is the CTO at Ursa & Tech Advisor for Start-Ups. He has expertise in building cloud-based machine learning and natural language processing services at early stage tech companies. He holds technical degrees from Cornell University and Stony Brook University.

Kennedy Behrman

INSTRUCTOR

Kennedy is a veteran consultant and author, specializing in architecting and implementing cloud solutions for early-stage startups. He is experienced in data engineering, data science, AWS solutions, and engineering management.

All Our Nanodegree Programs Include:

EXPERIENCED PROJECT REVIEWERS

REVIEWER SERVICES

- Personalized feedback & line by line code reviews
- 1600+ Reviewers with a 4.85/5 average rating
- 3 hour average project review turnaround time
- Unlimited submissions and feedback loops
- Practical tips and industry best practices
- Additional suggested resources to improve

TECHNICAL MENTOR SUPPORT

MENTORSHIP SERVICES

- Questions answered quickly by our team of technical mentors
- 1000+ Mentors with a 4.7/5 average rating
- Support for all your technical questions

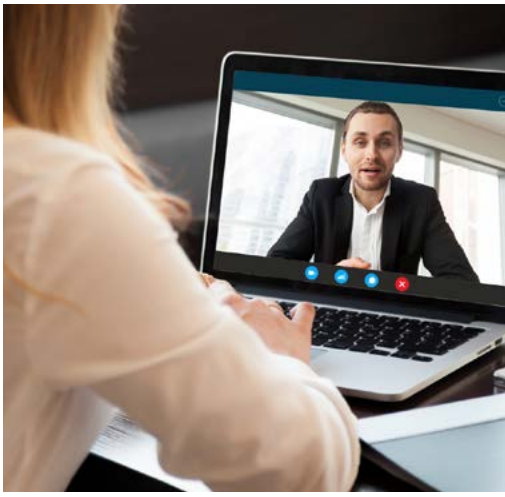
PERSONAL CAREER SERVICES

CAREER SUPPORT

- Resume support
- Github portfolio review
- LinkedIn profile optimization

Frequently Asked Questions

PROGRAM OVERVIEW



WHY SHOULD I ENROLL?

Becoming a software engineer is one of the best career moves you can make. Udacity built this Nanodegree program with input from leaders in the software industry to provide world-class Full Stack Web Development instruction that features code reviews and mentorship support throughout the program.

In the Full Stack Web Developer Nanodegree program, you will:

- Design and implement relational database systems to store and manage application data.
- Build dynamic software application backend systems using the Python programming language and the popular Flask application framework.
- Configure and deploy your applications to the cloud (Amazon Web Services)

WHAT JOBS WILL THIS PROGRAM PREPARE ME FOR?

Completion of the Full Stack Web Developer Nanodegree program will give you the tools needed to perform well in a variety of developer roles.

Some examples of job titles that align with your new skills are:
Software Engineer, Full Stack Python Application Developer, Back End Developer, Web Application Developer

HOW DO I KNOW IF THIS PROGRAM IS RIGHT FOR ME?

As a Full Stack Web Developer, you are the go-to person that companies rely on to build, support and maintain their web applications. Regardless of the platform, full stack web developers are in demand by nearly every company.If you are interested in building out the infrastructure that powers and supports the many web, desktop, mobile and integrated applications in the world, this program is the best way to get started.

WHAT IS THE DIFFERENCE BETWEEN THE FRONT END WEB DEVELOPER PROGRAM AND FULL STACK WEB DEVELOPER PROGRAM?

Web development generally fits into distinct concentrations, such as front end web development and full stack web development.

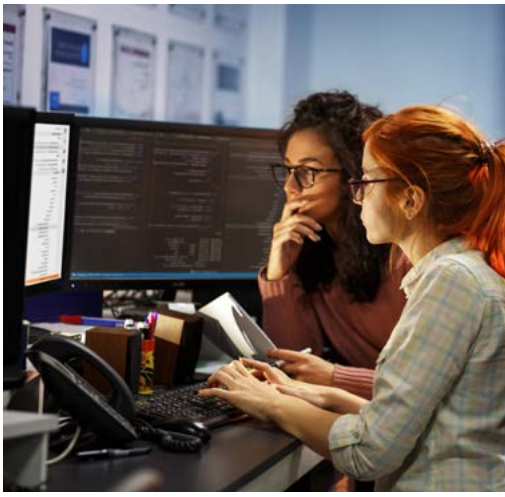
As a front end web developer, you'll build responsive, dynamic user interfaces on the web. You'll leverage your HTML, CSS, and JavaScript skills to manage all client-side scripting.

As a full stack web developer, you'll have an active hand in implementing relationship databases, configure and deploy your applications to the cloud, and build dynamic software application backend systems using the Python

FAQs Continued

programming language.

Whichever path you choose, you'll be building involved, engaging experiences on the web for your users!



ENROLLMENT AND ADMISSION

DO I NEED TO APPLY? WHAT ARE THE ADMISSION CRITERIA?

No. This Nanodegree program accepts all applicants regardless of experience and specific background.

WHAT ARE THE PREREQUISITES FOR ENROLLMENT?

Minimum Requirements:

- Beginner-level experience in Python. If you do not have this experience, check out our Intro to Programming Nanodegree program or Intro to Computer Science course.
- Experience building front-end web sites with HTML, CSS, and Javascript.
- Experience using Git for version control. If you do not have this experience, check out our How to Use Git and GitHub course.
- You are self-driven and motivated to learn. Participation in this program requires consistently meeting the deadlines, and devoting at least 10 hours per week to your work.
- You can communicate fluently and professionally in written and spoken English.
- You have access to a computer with a broadband connection, on which you'll install a professional code/text editor (ie. VSCode or Atom) as well as virtual machines (using VirtualBox and Vagrant).
- You are willing to contribute to the success of the program, including collaborating with fellow students, and giving us feedback on how we can improve.

Desirable Prior Experience::

- You've completed an object-oriented Python programming course.
- You've tried to build server-side applications in the past and want to learn how to do it at a professional level.

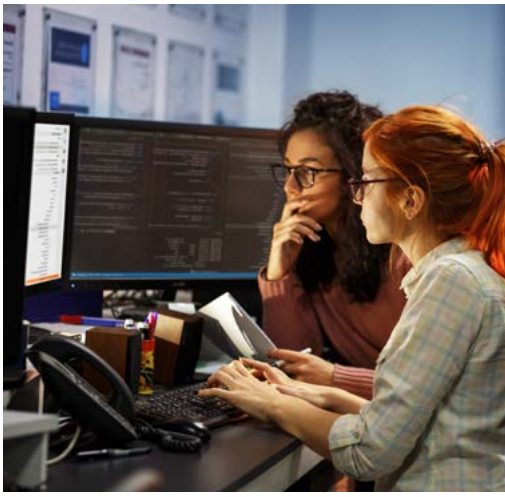
IF I DO NOT MEET THE REQUIREMENTS TO ENROLL, WHAT SHOULD I DO?

We have a number of Nanodegree programs and free courses that can help you prepare, including:

- Intro to Programming Nanodegree Program
- Data Structures and Algorithms Nanodegree Program
- Front End Web Developer Nanodegree Program

FAQs Continued

TUITION AND TERM OF PROGRAM



HOW IS THIS NANODEGREE PROGRAM STRUCTURED?

The Full Stack Web Developer Nanodegree program includes content and curriculum to support 5 (five) projects. We estimate that most students can complete the program in four (4) months working 5-10 hours per week.

Each project will be reviewed by the Udacity reviewer network. Feedback will be provided and if you do not pass the project, you will be asked to resubmit the project until it passes.

HOW LONG IS THIS NANODEGREE PROGRAM?

Access to this Nanodegree program runs for the length of time specified in the payment card above. If you do not graduate within that time period, you will continue learning with month to month payments. See the **Terms of Use** and **FAQs** for other policies regarding the terms of access to our Nanodegree programs.

SOFTWARE AND HARDWARE

WHAT SOFTWARE AND VERSIONS WILL I NEED IN THIS PROGRAM?

For this program, you will need a computer with a broadband internet connection, capable of hardware. Note: Most consumer computers on the market today meet these requirements. You will need administrative access to be able to install software on your computer. This program uses Python 3.7, PostgreSQL 11, SQLAlchemy, Flask 1.0, Docker and various Python packages. Students will need to be able to communicate fluently and professionally in written and spoken English.