Why Python Is the Fastest Way for High Schoolers to Get Into the Guts of AI

AI is everywhere now. But there's a big difference between **using AI** and **understanding how it works**.

If you're in high school (I have a nephew who is) and you're curious about what's happening "under the hood" (models, training, data, why it answers the way it does), **Python is the best starting point**. Not because it's trendy… but because most of the AI world is built around it.

Table of Contents

1. Why Python is the "default language" of AI
2. A beginner roadmap (made for high schoolers)
3. What languages AI "likes" (and where each fits)
4. How to deploy AI apps (so people can actually use them)
5. If you don't want Python: the "app-first" path
6. FAQ
7. Sources & links

Why Python is the "default language" of AI

Python wins in AI for a simple reason: **the tools are already built**. The most popular "learning and model" libraries are Python-first, and the tutorials, examples, and community support are massive.

That matters a lot when you're learning, because you don't want to spend 80% of your time fighting setup issues.

Python is readable (and that's a superpower)

When you're new, the code should feel like instructions, not like a puzzle. Python reads closer to plain English than most languages, which makes it easier to learn concepts like loops, functions, and data structures… which are basically the building blocks of everything.

Python is where the AI "ecosystem" lives

If you want to train models, experiment with data, or work through real tutorials, Python is the center of gravity. That's why it's such a strong "guts of AI" choice.

A beginner roadmap (made for high schoolers)



This is a realistic path that doesn't assume you already know a ton.

Step 1: Learn just enough Python to be dangerous

- Variables, loops, functions
- Lists/dictionaries
- Reading/writing files
- Basic debugging (how to find what broke)

Tip: Don't spend months "studying" before building anything. Learn a little ? build a tiny thing ? repeat.

Step 2: Build tiny AI-ish projects (no supercomputer required)

- **Spam vs. not spam classifier** (text classification)
- **Image classifier** using a beginner notebook tutorial
- **Simple "recommendation" project** (rank items based on rules + data)

Step 3: Learn one "real" ML library

If you want a clean on-ramp, start with **scikit-learn** for classic machine learning (great for learning the basics without getting overwhelmed). Then move to deep learning (PyTorch or TensorFlow) when you're ready.

What languages AI "likes" (and where each fits)

People argue about this online a lot, so here's the simplest honest answer:

Python: training, experimenting, data, and "real ML work"

Most learning resources, research code, and model training workflows assume Python. If you want to understand how models behave (and why), Python is the clearest path.

JavaScript/TypeScript: building the actual apps people use

A lot of AI products are basically:

- a website UI (often JavaScript/TypeScript),
- a server/API (could be Node.js or Python),
- and an AI model/API behind it.

If you like designing experiences and shipping real apps, JS/TS is a great lane.

SQL: the quiet MVP

If your AI app touches real data, you'll eventually see SQL. It's how tons of systems store and query information.

C/C++ (and sometimes Rust/Go): performance and infrastructure

You may not start here, but these show up when speed and systems-level work matter. Think: "make it faster" or "make it run everywhere."

How to deploy AI apps (so people can actually use them)

Lots of projects die in this sad place: **"It works on my laptop."**

Deployment is just the process of turning your project into something other people can open and use.

Deployment concepts to learn (even if you're a beginner)

- **Git + GitHub**: your project lives in a repo, and deployment platforms connect to it
- **Environment variables**: where secrets (API keys) should live (not inside your code)
- **Dependencies**: a clean list of what your project needs installed
- **Containers (Docker)**: a standard "box" your app runs in (not always required, but super useful)

Fast way to deploy a Python AI app (two beginner-friendly options)

Option A: Streamlit (great for simple apps)
You can build a UI in pure Python and deploy it with minimal fuss. It's a great way to show a project to friends, teachers, or even a college admissions portfolio.

Option B: Gradio (great for demos around a model)
Gradio makes it ridiculously easy to wrap a model or function with an interactive interface and share it.

When you need a "real API": FastAPI

If you want an app with a front-end (JS/TS) calling a back-end, a Python API framework like FastAPI is a common move. You write endpoints, your front-end calls them, and your AI runs on the server.

Beginner-friendly deployment targets:

- **Render** (simple web service deployment)
- **Railway** (good docs, multiple deployment options)

One more real-world note: if you deploy anything using an AI API, learn how to keep API keys secret. Keys should be stored server-side and loaded securely, not baked into front-end code.

If you don't want Python: the "app-first" path

Maybe you don't care about training models. You just want to **build useful apps** (and ship them fast). That's totally valid.

In that case, you can focus on:

- **JavaScript/TypeScript** (front-end + back-end)
- **Using AI APIs** (instead of training your own model)
- **Deployment platforms** that publish your app from a Git repo

A very common "app-first" stack looks like this:

- **Next.js** (build the website + server routes in one project)
- **Vercel** or **Netlify** (deploy from GitHub)
- **OpenAI (or similar) SDK** for the AI features

That approach lets you create real projects people can use, without going deep into model training right away. And honestly… once you've shipped a couple apps, learning Python later becomes way easier because you'll already "think like a developer."

FAQs About How Teens and Get into AI

Do I need to be “good at math” to start AI?

No. You’ll eventually run into math concepts, but you can start building projects now. The math gets easier once you have real context for why it matters.

Should I learn Python or JavaScript first?

If you want the **guts of AI**, start with Python. If you want to **ship apps** fast, JavaScript/TypeScript is a great first language. Either path is valid.

Do I need a powerful computer?

Not at the start. You can learn and build plenty with beginner datasets and browser tools (like notebook-based tutorials). When you outgrow your laptop, you’ll know exactly why.

What should my first “portfolio” AI project be?

A small app with a clear purpose beats a giant confusing one. Example: a simple web page that takes text input and returns a useful structured result (summary, categories, study flashcards, etc.).

Sources & Links to Get Started

- Python Official Tutorial
- Python.org: Getting Started
- Python Wiki: Beginner’s Guide
- scikit-learn: Getting Started
- PyTorch Tutorials: Learn the Basics
- TensorFlow Beginner Quickstart
- Gradio Quickstart
- Streamlit: Deploy Your App (Community Cloud)
- FastAPI Tutorial
- FastAPI Deployment
- Render: Deploy a FastAPI App
- Railway: Deploy a FastAPI App
- Docker: Get Started
- OpenAI API: Quickstart
- OpenAI API: Official Libraries
- OpenAI API: Key Safety Notes
- Stack Overflow Developer Survey 2025 (Technology)
- Vercel: Next.js on Vercel
- Netlify: Next.js Overview
- Hugging Face Spaces: Streamlit Notes

- Hugging Face Spaces: Gradio

Original article: https://tonyherman.com/python-and-ai-for-high-schoolers/
PDF version: https://tonyherman.com/python-and-ai-for-high-schoolers/?apa_pdf=1

Special Offer for Readers

1,000+ Channels • On-Demand Movies • **5 Devices**

$69.99/mo (~$2.33/day)

Try it for a month and see if you like it, then switch.

Start Your Trial

Click to learn more about CUE Broadcast

Tip: Get 3 friends of family to sign up and you it for free.