ChatGPT

100 Python Projects to Improve Your Skills

Working on hands-on projects is an excellent way to practice coding and learn by doing. As Dataquest notes, *“Completing Python projects is the ultimate way to learn the language”* [1], and you’ll *“learn more by building than by reading alone”* [2]. GeeksforGeeks similarly observes that projects are *“a fun way to learn and practice coding… They help you build practical skills like making games, automating tasks, analyzing data, and creating web apps”* [3].

The following 100 project ideas span domains such as automation, web development, data science, game development, and APIs, and each is labeled with an estimated difficulty (Beginner, Intermediate, Advanced). They range from short scripts to long-term builds, providing a structured path from simple to complex tasks.

1. **Personal Portfolio Website** (Beginner) — Create a static personal website using Flask to showcase your projects and resume. Focus on HTML/CSS templates, Flask routing, and deploying the site (e.g., on Heroku or GitHub Pages).
2. **Blog Engine with Django** (Intermediate) — Develop a multi-user blog with Django, including features like user authentication, post creation/editing, and comments. This teaches you about databases (models), Django’s ORM, templates, and form handling.
3. **Task Manager (To-Do List)** (Intermediate) — Build a to-do list app using Django where users can create, update, and delete tasks. Learn about CRUD operations, URL routing, and organizing views and templates.
4. **RESTful API for a To-Do App** (Intermediate) — Create a Flask or Django REST Framework API for managing tasks (create/read/update/delete). Use JSON for data exchange and optionally connect it to a simple frontend or use Postman for testing.
5. **Real-Time Chat Application** (Advanced) — Use Django Channels or Flask-SocketIO to build a live chat app. Implement WebSocket communication so multiple users can send and receive messages in real time.
6. **E-commerce Website** (Advanced) — Develop a full-featured online store (with Django or Flask) that handles product listings, shopping carts, orders, and payments. Integrate a payment API (like Stripe or PayPal) and implement user authentication and an admin interface.
7. **URL Shortener Service** (Intermediate) — Build a web app that generates short URLs for long links and redirects to the original URL. Use Flask or Django, a database to map short codes to URLs, and ensure you handle duplicates and collisions.
8. **Weather Dashboard** (Intermediate) — Create a web app that fetches weather data from a public API (e.g., OpenWeatherMap) and displays current conditions and forecast for a given city. Use Flask and a charting library or maps to visualize the data.
9. **Discussion Forum** (Intermediate) — Implement a forum where users can post topics and comments (like a simple Reddit or StackOverflow). Use Django for user accounts, thread creation, replies, and up/down voting or rating.
10. **Social Media Aggregator** (Intermediate) — Build a dashboard that pulls in your feeds from social networks (Twitter, Instagram, etc.) using their APIs and displays them in one place. Practice working with multiple APIs and handling JSON data.

11. **Quiz Website** (Intermediate) — Develop an online quiz platform (Flask/Django) where users can take quizzes on various topics. Include features like timed questions, scoring, and storing results in a database.
12. **Online Poll/Survey Tool** (Intermediate) — Create a web app to conduct polls or surveys. Users can create polls with multiple choices, vote on them, and view results. This involves forms, data models, and rendering charts of vote counts.
13. **Online Code Editor** (Advanced) — Build a simple web-based code editor in Python. Allow users to write and run Python code in the browser (taking care to sandbox execution for security). You might use a library like Brython or implement server-side execution in a Docker container.
14. **Microblogging Platform** (Advanced) — Implement a Twitter-like site where users post short messages, follow others, and see a live feed. Includes user profiles, image uploads, and real-time updates.
15. **Inventory Management System** (Intermediate) — Create a web app for tracking products in a store or warehouse. Include features for adding/editing items, updating stock levels, and recording sales. Use a relational database (SQLite/PostgreSQL) for data storage.
16. **Booking System** (Intermediate) — Develop a booking application (e.g., for hotels, flights, or events) where users can search for availability and reserve slots. Handle date/time inputs, form submissions, and prevent double bookings.
17. **Stock Trading Simulator** (Advanced) — Build a dashboard that displays real stock data (using a finance API like yFinance) and lets users simulate buying/selling stocks with virtual money. Include interactive price charts and portfolio tracking.
18. **Event Calendar App** (Intermediate) — Make a web app to schedule events (meetings, birthdays, etc.) on a calendar view. Users can create events on specific dates and view them in a monthly or weekly calendar interface.
19. **User Authentication Service** (Intermediate) — Create a standalone authentication system (Flask or Django) that supports user registration, login, password reset, and email verification. Consider using JWT tokens or OAuth for session management.
20. **YouTube Video Downloader** (Intermediate) — Use the YouTube API or a library like `pytube` to build a tool that downloads videos from YouTube given a URL. Add options for format and resolution selection.
21. **File Organizer Script** (Beginner) — Write a script that scans a directory and moves files into subfolders based on file type or date (e.g., move all `.jpg` to an "Images" folder). This automates file management on your computer.
22. **Automated Email Sender** (Intermediate) — Create a Python script using `smtplib` that sends emails automatically. For example, send scheduled reminders or personalized emails (reading recipient addresses from a CSV or database).
23. **Job Listing Scraper** (Intermediate) — Build a web scraper (using BeautifulSoup or Scrapy) to collect job postings from sites like Indeed or LinkedIn (if allowed). Extract details like job title, company, and location, and save them to a file or database.
24. **Invoice Generator** (Intermediate) — Automate invoice creation by reading order data (items, prices, customer info) from a CSV/JSON file and generating a PDF invoice using a library like ReportLab or WeasyPrint.
25. **Bulk Image Resizer** (Beginner) — Use Pillow (PIL) to create a script that resizes or compresses all images in a folder to a specified dimension or quality. Useful for optimizing a photo collection.
26. **Daily Backup Script** (Intermediate) — Write a script that zips important files or directories and saves the backup with a timestamp. Optionally automate it to run daily (e.g., with cron or Task Scheduler) or upload backups to a cloud service.

27. **PDF Data Extractor** (Intermediate) — Use a library like PyPDF2 or pdfminer.six to extract text or tables from PDF documents. For example, parse receipts or reports and export key data to a spreadsheet.
28. **Twitter Posting Bot** (Intermediate) — Use the Twitter API (via Tweepy) to create a bot that automatically posts tweets or retweets. For example, schedule daily inspirational quotes or news updates.
29. **Telegram Chatbot** (Intermediate) — Build a Telegram bot (using python-telegram-bot) that responds to user commands. It could provide weather information, jokes, or news summaries on request.
30. **Website Monitor** (Intermediate) — Write a script that periodically checks if a list of websites are up (using `requests` or `urllib`). Log response times and send an alert (email or SMS via Twilio) if any site is down.
31. **Automated Form Filler** (Intermediate) — Use Selenium to automate filling out web forms. For example, auto-login to a site, post a comment, or submit survey data, learning to control the browser programmatically.
32. **CLI Expense Tracker** (Beginner) — Develop a simple command-line program to log daily expenses and categorize them. Store the data in a CSV or SQLite database and allow the user to view monthly summary reports.
33. **Stock Price Alert** (Intermediate) — Use a financial API (like Alpha Vantage) to get real-time stock prices. Write a script that checks for price thresholds (e.g., if a stock goes above/below a target) and sends notifications (email or SMS).
34. **RSS Feed Aggregator** (Intermediate) — Create a script that reads multiple RSS feeds (using `feedparser`) and compiles new articles into a single HTML or CSV file. Good for monitoring news or blog updates in one place.
35. **Bulk Text Translator** (Intermediate) — Use a translation API (like Google Translate API or the `googletrans` library) to translate all text files in a folder to another language. Practice working with external APIs and file I/O.
36. **Cloud Photo Backup Tool** (Intermediate) — Automate backing up a folder of photos to a cloud service. For example, use the Dropbox or Google Drive API to upload new images from a local directory to your cloud storage.
37. **Data Logger** (Intermediate) — Simulate a sensor-reading script that logs data (like temperature or CPU usage) with timestamps into a file or database. This teaches file I/O and how to schedule recurring tasks (using `schedule` or cron).
38. **Titanic Survival Predictor** (Intermediate) — Analyze the classic Titanic dataset (from Kaggle) using pandas and scikit-learn. Build a machine learning model to predict passenger survival based on features like age, class, and fare.
39. **Iris Flower Classifier** (Beginner) — Use the Iris dataset with scikit-learn to classify iris plants into species (setosa, versicolor, virginica). This is a small dataset that teaches basic data processing and model training.
40. **House Price Predictor** (Intermediate) — Use a regression model (scikit-learn) on a housing dataset (like the Boston housing data or a Kaggle dataset) to predict house prices. Practice feature engineering and evaluating model accuracy.
41. **Email Spam Filter** (Intermediate) — Build a spam detection system using techniques like Naive Bayes or logistic regression. Train it on a dataset of labeled emails (ham/spam) and test its accuracy on unseen messages.

42. **Stock Price Forecasting** (Advanced) — Apply time-series forecasting (e.g., ARIMA or LSTM neural networks) to historical stock price data. Predict future prices or trends and evaluate the model's performance.
43. **Image Classifier** (Intermediate) — Use TensorFlow or PyTorch to build a convolutional neural network that classifies images (e.g., cats vs. dogs, or various objects). Learn about data augmentation and model evaluation.
44. **Tweet Sentiment Analysis** (Intermediate) — Collect tweets on a topic (using Tweepy) and analyze their sentiment (positive/negative) using NLTK, TextBlob, or a pretrained transformer. Visualize sentiment trends over time.
45. **Seq2Seq Chatbot** (Advanced) — Implement a sequence-to-sequence (encoder-decoder) neural network (with attention) to build a chatbot. Train it on conversational data and test its ability to generate human-like responses.
46. **Voice Assistant** (Intermediate) — Create a simple voice assistant that listens to microphone input (using `SpeechRecognition`) and can perform tasks like telling the time, fetching Wikipedia summaries, or playing a YouTube video.
47. **Game AI Agent** (Intermediate) — Write an AI agent to play a simple game (like Tic-Tac-Toe or Connect Four) using algorithms like Minimax. Optionally add alpha-beta pruning or try making it play a higher-level game (chess, checkers).
48. **Recommendation System** (Intermediate) — Build a recommender (for movies, books, etc.) using collaborative filtering or content-based filtering. Use a library like Surprise or implement matrix factorization from scratch.
49. **Optical Character Recognition (OCR)** (Intermediate) — Use an OCR library (such as `pytesseract`) to extract text from images. For example, take scanned receipts or photos of documents and convert them into editable text.
50. **Face Recognition System** (Advanced) — Use OpenCV and a pre-trained model to detect and recognize faces in images or live video. This involves learning about face detection (Haar cascades or deep learning models) and face embeddings.
51. **Interactive Data Dashboard** (Intermediate) — Use Plotly Dash or Streamlit to create an interactive dashboard. Load a dataset (like global COVID-19 statistics) and display charts, maps, and user-selectable filters.
52. **Neural Style Transfer** (Advanced) — Apply neural style transfer to blend the style of one image (e.g., a famous painting) with the content of another image. Implement this using TensorFlow or PyTorch with a pretrained model.
53. **Object Detection App** (Advanced) — Use a model like YOLO or SSD to detect and label multiple objects in images or live video (e.g., detect cars, people, animals). Practice computer vision and deep learning.
54. **Speech-to-Text Transcriber** (Intermediate) — Use an API (Google Speech-to-Text) or Python's `SpeechRecognition` to convert audio recordings into text. Handle different audio formats and noise levels.
55. **Language Translator** (Intermediate) — Implement a simple translation tool. You could train a small neural network on bilingual sentence pairs or use the Google Translate API to translate text between languages.
56. **Genetic Algorithm Solver** (Intermediate) — Implement a genetic algorithm to solve an optimization problem (for example, the Traveling Salesman Problem or scheduling). Learn about selection, crossover, and mutation operators.

57. **Cancer Detection from Medical Images** (Advanced) — Use deep learning (e.g., CNNs) to classify medical images (like X-rays or MRIs) as cancerous or healthy. This project requires careful preprocessing and evaluation due to ethical considerations.
58. **Handwritten Digit Recognizer** (Intermediate) — Use the MNIST dataset to train a neural network that recognizes handwritten digits (0-9). Practice with Keras or PyTorch on image data and evaluate model accuracy.
59. **Generative Adversarial Network (GAN)** (Advanced) — Build a GAN to generate new images (e.g., faces or fashion items). Implement the generator and discriminator networks and train them against each other.

60. **Social Media Sentiment Bot** (Advanced) — Create a bot that fetches social media posts on a topic, analyzes sentiment, and posts a summary of overall sentiment or trends. Combines API access, NLP, and automated posting.

61. **Text-Based Adventure Game** (Beginner) — Write a console-based adventure game (like a dungeon crawler) where the player explores rooms, picks up items, and solves puzzles through text commands.

62. **Tic-Tac-Toe with AI** (Intermediate) — Create a Tic-Tac-Toe game in the console or with a GUI. Allow the player to play against the computer, which uses the Minimax algorithm for perfect play.
63. **Snake Game** (Intermediate) — Use Pygame to develop the classic Snake game. The player controls a snake that grows longer when eating food; the game ends if the snake collides with itself or the wall.
64. **Breakout (Brick Breaker) Game** (Intermediate) — Build a game where a ball bounces around and the player controls a paddle to break bricks (like Arkanoid). Use Pygame for rendering and collision detection.
65. **Tower Defense Game** (Advanced) — Create a tower-defense style game using Pygame or another framework. Enemies walk along a path, and the player places towers that shoot projectiles to defend a base.
66. **Multiplayer Quiz Game** (Intermediate) — Develop a quiz game where multiple players can answer questions (over a network or local) and compete. Keep score and handle turn-taking or simultaneous answering.
67. **2D Platformer** (Advanced) — Build a side-scrolling platform game (like Mario) using Pygame or Arcade. Implement player movement, jumping physics, platforms, and enemy characters.
68. **Maze Runner Game** (Intermediate) — Generate a random maze and build a game where the player navigates out. Optionally, write a solver that finds the shortest path through the maze.
69. **Space Invaders Clone** (Intermediate) — Recreate the classic Space Invaders game using Pygame. Include a player ship at the bottom that shoots up at descending aliens.
70. **Memory Matching Game** (Beginner) — Create a card-matching game where cards are laid face-down and the player flips two at a time trying to match pairs. Implement with a simple GUI (Tkinter or Pygame).
71. **Pong Game** (Beginner) — Develop the classic Pong game for two players (or player vs. computer) using Pygame. Each paddle is controlled by a player, and a ball bounces between them.
72. **Blackjack Card Game** (Intermediate) — Simulate the Blackjack (21) card game with a GUI or console interface. Implement dealing cards, player choices (hit/stand), and betting logic.
73. **Physics Simulation** (Intermediate) — Use Pygame or matplotlib to simulate physics, such as bouncing balls with gravity and collision detection or a swinging pendulum with damping.

74. **Chess Game** (Advanced) — Create a full chess game with all pieces and rules. Implement move validation and game logic. Optionally add an AI opponent using Minimax or a third-party chess library.

75. **Augmented Reality Demo** (Advanced) — Use OpenCV to overlay virtual objects onto a live camera feed. For example, detect a marker in the video and draw a 3D cube or character on it.

76. **Sudoku Solver** (Intermediate) — Write a program that solves Sudoku puzzles using backtracking or constraint satisfaction. Input a 9×9 grid (with zeros or blanks for empty cells) and output the solved puzzle.

77. **Web Crawler / Search Engine** (Intermediate) — Build a simple web crawler that starts from a seed URL, follows links, and indexes pages. Allow basic keyword search over the indexed pages.
78. **Maze Solver** (Intermediate) — Given a text or image representation of a maze, write an algorithm (BFS, DFS, or A*) that finds a path from start to finish.
79. **Prime Number Finder** (Beginner) — Write a program to list all prime numbers up to a large N or check if a given number is prime. Implement a basic algorithm like trial division or the Sieve of Eratosthenes.
80. **Sorting Algorithm Visualizer** (Intermediate) — Create a visualization tool (using Tkinter or Pygame) that shows how different sorting algorithms (bubble sort, quicksort, etc.) reorder a list step by step.
81. **Caesar Cipher Tool** (Beginner) — Implement the Caesar cipher for encryption and decryption of text. Allow the user to input a message and a shift value to encode/decode.
82. **AES Encryption Utility** (Intermediate) — Build a command-line tool to encrypt and decrypt files using AES (Advanced Encryption Standard). Use a library like PyCryptodome and handle keys securely.
83. **Password Strength Checker** (Intermediate) — Create a program that rates the strength of a password. Check criteria like length, use of uppercase/lowercase letters, numbers, symbols, and absence of dictionary words.
84. **CPU Scheduling Simulator** (Advanced) — Simulate operating system CPU scheduling algorithms (Round Robin, Priority, etc.). Input a list of processes with burst times and simulate scheduling to compute turnaround and wait times.
85. **RPG Battle Simulator** (Intermediate) — Write a turn-based battle simulator (text or GUI) for a simple RPG. Include player and enemy stats (HP, attack, defense), and actions (attack, heal, use potion).
86. **Supply-Demand Optimizer** (Advanced) — Formulate a linear programming problem (e.g., maximize profit subject to resource constraints) and solve it using a solver like PuLP. For example, optimize production mix or transportation logistics.
87. **Math Quiz Generator** (Beginner) — Automatically generate random arithmetic problems (addition, subtraction, multiplication, division) for a user to solve. Keep track of score and give feedback.
88. **Traveling Salesman Heuristic Solver** (Intermediate) — Implement a heuristic (like nearest neighbor or simulated annealing) for the Traveling Salesman Problem to find an approximate shortest route through a set of cities.
89. **Neural Network from Scratch** (Advanced) — Implement a simple neural network with one or two layers without using high-level libraries. Code the forward propagation and backpropagation (e.g., for binary classification on synthetic data).
90. **Webcam Filter Application** (Intermediate) — Use OpenCV to capture video from your webcam and apply real-time filters (e.g., grayscale, blur, edge detection). Display the filtered video in a window.
91. **Chatbot with GPT-3.5 API** (Advanced) — Build a chatbot interface that uses OpenAI's GPT-3.5 (or similar) API to generate responses. Handle conversation context and API rate limits.

92. **Calendar Notification App** (Intermediate) — Create a program that reads events from a calendar file or API and sends daily email/SMS reminders of upcoming events (using an email service or Twilio).
93. **Music Player GUI** (Intermediate) — Build a desktop music player using Tkinter or PyQt. Include play/pause controls, a playlist, and the ability to load and play local MP3 files using a library like Pygame's mixer.
94. **Inventory/Stock CLI Tool** (Intermediate) — Develop a command-line inventory management system to track product stock and sales. Store data in a database or CSV and support operations like adding new products, updating quantities, and generating reports.
95. **Air Quality Dashboard** (Intermediate) — Use an API (like OpenAQ) to fetch air quality index (AQI) data for your location and display it in a GUI or web app. Include charts of pollutant levels over time.
96. **News Article Summarizer** (Advanced) — Use an NLP approach (or a pretrained transformer model like BERT) to generate summaries of news articles. Fetch articles (via an API or web scrape) and display concise summaries.
97. **Server Log Analyzer** (Intermediate) — Write a tool to parse web server logs (Apache/Nginx). Extract useful statistics (most visited pages, error counts, peak traffic times) and output a summary report.
98. **Machine Learning Model Web App** (Advanced) — Take a trained ML model (e.g., a skin cancer image classifier) and deploy it with Flask or Django. Build a web interface to upload input (like an image) and display the model's prediction.
99. **NASA APOD Viewer** (Beginner) — Use NASA's public API to fetch and display the Astronomy Picture of the Day. Show the image and its description in a simple GUI (Tkinter) or web page (Flask).
100. **Home Automation with IoT** (Advanced) — Connect to IoT devices in your home using Python. For example, control smart lights (Philips Hue API or MQTT), read sensor data (temperature, motion), and automate tasks like turning lights on/off on a schedule.

Each project above is tagged with an estimated difficulty and can be scaled in scope. Beginners might start with simple scripts or games, while intermediates can tackle web apps and basic ML tasks. Advanced projects often integrate multiple components (e.g., APIs, GUIs, AI) and may require weeks of work. Regardless of level, building these projects will deepen your understanding of Python in practical contexts [3] [2].

Sources: These project ideas are inspired by educational resources on Python learning and project-based practice [1] [3].

---

[1] [2] Python Projects for Beginners: 60+ Ideas to Build Your Portfolio – Dataquest
https://www.dataquest.io/blog/python-projects-for-beginners/

[3] Python Projects – Beginner to Advanced | GeeksforGeeks
https://www.geeksforgeeks.org/python-projects-beginner-to-advanced/