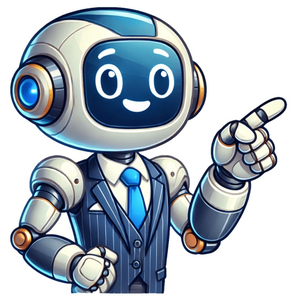
Continue



The best way to learn a new programming language is to build projects with it. I have created a list of 25 beginner friendly project tutorials in Python. My advice for tutorials would be to watch the video, build the project, break it apart and rebuild it your own way. Experiment with adding new features or using different methods. That will test if you have really learned the concepts or not. You can click on any of the projects listed below to jump to that section of the article. If you are not familiar with the basics of Python, then I would suggest watching this beginner freeCodeCamp Python tutorial. In this Kylie Ying tutorial, you will learn how to get input from the user, work with f-strings, and see your results printed to the console. This is a great starter project to get comfortable doing string concatenation in Python. In this Kylie Ying tutorial, you will learn how to work with Python's random module, build functions, work with while loops and conditionals, and get user input. In this Kylie Ying tutorial, you will build a guessing game where the computer has to guess the correct number. You will work with Python's random module, build functions, work with while loops and conditionals, and get user input. In this Kylie Ying tutorial, you will work with random.choice(), if statements, and getting user input. This is a great project to help you build on the fundamentals like conditionals and functions. In this Kylie Ying tutorial, you will learn how to work with dictionaries, lists, and nested if statements. You will also learn how to work with the string and random Python modules. In this Code With Tomi tutorial, you will learn how to build a countdown timer using the time Python module. This is a great beginner project to get you used to working with while loops in Python. In this Code With Tomi tutorial, you will learn how to build a random password generator. You will collect data from the user on the number of passwords and their lengths and output a collection of passwords with random characters. This project will give you more practice working with for loops and the random Python module. In this Code With Tomi tutorial, you will learn how to create your own QR codes and encode/decode information from them. This project uses the qrcode library. This is a great project for beginners to get comfortable working with and installing different Python modules. In this Kylie Ying tutorial, you will learn how to build a tic-tac-toe game with various players in the command line. You will learn how to work with Python's time and math modules as well as get continual practice with nested if statements. In this Kylie Ying tutorial, you will learn how to build a tic-tac-toe game where the computer never loses. This project utilizes the minimax algorithm which is a recursive algorithm used for decision making. In this Kylie Ying tutorial, you will learn how to implement the divide and conquer algorithm called binary search. This is a common searching algorithm which comes up in job interviews, which is why it is important to know how to implement it in code. In this Kylie Ying tutorial, you will build the classic minesweeper game in the command line. This project focuses on recursion and classes. In this Kylie Ying tutorial, you will learn how to build a sudoku solver which utilizes the backtracking technique. Backtracking is a recursive technique that searches for every possible combination to help solve the problem. In this Kylie Ying tutorial, you will learn how to create an image filter and change the contrast, brightness, and blur of images. Before starting the project, you will need to download the starter files. Markov Chain Text Composer Python Project In this Kylie Ying tutorial, you will learn about the Markov chain graph model and how it can be applied the relationship of song lyrics. This project is a great introduction into artificial intelligence in Python. In this Christian Thompson tutorial, you will learn how to recreate the classic pong game in Python. You will be working with the os and turtle Python modules which are great for creating graphics for games. In this Tech with Tim tutorial, you will learn how to recreate the classic snake game in Python. This project uses Object-oriented programming and Pygame which is a popular Python module for creating games. In this Keith Galli tutorial, you will learn how to build the classic connect four game. This project utilizes the numpy, math, pygame and sys Python modules. This project is great if you have already built some smaller beginner Python projects. But if you haven't built any Python projects, then I would highly suggest starting with one of the earlier projects on the list and working your way up to this one. In this Tech with Tim tutorial, you will learn how to recreate the classic Tetris game. This project utilizes Pygame and is great for beginner developers to take their skills to the next level. In this Tech with Tim tutorial, you will learn how to build an online multiplayer game where you can play with anyone around the world. This project is a great introduction to working with sockets, networking, and Pygame. In this Code With Tomi tutorial, you will learn how to ask for user input for a GitHub user link and output the profile image link through web scraping. Web scraping is a technique that collects data from a web page. In this Code With Tomi tutorial, you will learn how to build a program that can go into any folder on your computer and rename all of the files based on the conditions set in your Python code. In this Code With Tomi tutorial, you will learn how to build a program that collects user data on a specific location and outputs the weather details of that provided location. This is a great project to start learning how to get data from API's. In this Beau Carnes tutorial, you will learn how to build your own bot that works in Discord which is a platform where people can come together and chat online. This project will teach you how to work with the Discord API and Replit IDE. After this video was released, Replit changed how you can store your environments variables in your program. Please read through this tutorial on how to properly store environment variables in Replit. In this buildwithpython tutorial, you will learn how to build a space invaders game using Pygame. You will learn a lot of basics in game development like game loops, collision detection, key press events, and more. 1. Natural language processing (NLP) project: Develop a program that can analyze and interpret human language, including sentiment analysis, named entity recognition, and text classification.2. Autonomous drone navigation: Build a program that uses computer vision and machine learning to enable autonomous drone navigation.3. Speech recognition: Create a program that can recognize and transcribe human speech using deep learning techniques.4. Recommendation system: Develop a program that recommends products, services, or content to users based on their preferences and behavior.5. Facial recognition: Build a program that can recognize and identify individuals in images or videos using computer vision techniques.6. Fraud detection: Create a program that can detect fraudulent activity in financial transactions using machine learning algorithms.7. Object detection: Develop a program that can detect and classify objects in images or videos using deep learning models.8. Chatbot with emotion recognition: Build a conversational AI that can understand and respond to human emotions in a natural and empathetic way.9. Autonomous vehicle control: Develop a program that uses machine learning and computer vision to enable autonomous vehicle control.10. Stock trading algorithm: Create a program that uses machine learning and data analysis to predict stock prices and make profitable trades.11. Music generation: Build a program that uses machine learning algorithms to generate original music compositions.12. Computer vision project: Develop a program that can recognize and classify images or videos in real-time, such as detecting traffic signs or identifying objects in a manufacturing plant.13. Blockchain application: Build a blockchain-based application that enables secure and transparent transactions.14. Reinforcement learning project: Create a program that uses reinforcement learning algorithms to enable decision-making and control in complex environments, such as robotics or game-playing AI.Note: It is outside the scope of this article to create an advanced project for readers! Learning Python does not have to be all theory and syntax drills. In fact, one of the best ways to master Python is by building fun python projects that challenge your creativity and problem-solving skills. Whether you are a beginner learning to code, a high school student preparing for computer science courses, or a college student exploring new interests, working on fun python projects makes the learning process exciting, practical, and memorable.From building interactive games and personality quizzes to designing automation scripts, creative calculators, or even AI-based tools, fun python projects allow you to apply what you learn in engaging ways. These projects help solidify key concepts like loops, conditionals, data structures, and APIs. Best of all, they are perfect additions to your portfolio, showing initiative and technical ability. This kind of work stands out on college applications, internship resumes, and job interviews. By consistently working on fun python projects, you build both your confidence and coding fluency while enjoying the process. Fun python projects are creative and engaging ways for high school students to explore coding while enjoying the process. These projects turn learning into an adventure and are a great way to develop real programming skills while working on something meaningful or entertaining.A fun python project is one that :Sparks curiosity or aligns with your personal interestsIs creative, playful, or interactiveChallenges your logic without overwhelming youProduces something you can share, show off, or actually useFun python projects for high school students might include games, mini apps, quizzes, automation tools, or even AI-powered bots. These kinds of projects help break the monotony of theory-based learning, making programming feel more like solving a puzzle than completing a chore. Whether you are new to Python or looking to grow your skills, working on fun python projects is one of the best ways to stay motivated and improve. Create a game where the computer generates a random number, and the player guesses it with hints like "too high" or "too low." Perfect for learning basic Python logic.Skills Learned: Random module, loops, conditionalsDifficulty: BeginnerLearn more about Number Guessing Game.2. Tic-Tac-ToeBuild a two-player Tic-Tac-Toe game with a graphical interface using Tkinter or Pygame, teaching game logic and UI design.Skills Learned: OOP, GUI development, event handlingDifficulty: BeginnerLearn more about Tic-Tac-Toe.3. HangmanDevelop a word-guessing game where players guess letters to complete a word, with a limited number of attempts.Skills Learned: String manipulation, loops, TkinterDifficulty: BeginnerLearn more about Hangman.4. Mad Libs GeneratorCreate a story generator that prompts users for words (nouns, verbs, etc.) and inserts them into a funny narrative.Skills Learned: String concatenation, user inputDifficulty: BeginnerLearn more about Mad Libs Generator.5. Simple CalculatorBuild a calculator for basic operations (add, subtract, multiply, divide) with a console or Tkinter interface.Skills Learned: Functions, user input, basic mathDifficulty: BeginnerLearn more about Simple Calculator.6. Dice RollerSimulate rolling dice with random number generation, customizable for different dice types (e.g., D6, D20).Skills Learned: Random module, loopsDifficulty: BeginnerLearn more about Dice Roller.7. Rock, Paper, ScissorsDevelop a game where players compete against the computer using random choices and simple logic.Skills Learned: Random module, conditionalsDifficulty: BeginnerLearn more about Rock, Paper, Scissors.8. Turtle Graphics SquareUse the Turtle module to draw a square or other shapes, introducing graphical programming basics.Skills Learned: Turtle graphics, loopsDifficulty: BeginnerLearn more about Turtle Graphics.9. Password GeneratorCreate a tool to generate random, secure passwords with customizable length and characters.Skills Learned: String module, random moduleDifficulty: BeginnerLearn more about Password Generator.10. Countdown TimerBuild a timer that counts down from a user-specified time, useful for tasks like studying or cooking.Skills Learned: Time module, loopsDifficulty: BeginnerLearn more about Countdown Timer.11. Digital ClockCreate a real-time clock displaying hours, minutes, and seconds using Tkinter for a GUI.Skills Learned: Tkinter, time moduleDifficulty: BeginnerLearn more about Digital Clock.12. Email SlicerExtract the username and domain from an email address using string manipulation.Skills Learned: String manipulation, user inputDifficulty: BeginnerLearn more about Email Slicer.13. Weather AppFetch real-time weather data using an API and display it in a console or GUI interface.Skills Learned: APIs, requests module, GUIDifficulty: IntermediateLearn more about Weather App.14. Snake GameRecreate the classic Snake game using Pygame, where the snake grows by eating food while avoiding collisions.Skills Learned: Pygame, game logic, event handlingDifficulty: IntermediateLearn more about Snake Game.15. Pong GameBuild the classic Pong game with Pygame, featuring paddles and a bouncing ball.Skills Learned: Pygame, collision detection, game loopDifficulty: IntermediateLearn more about Pong Game.16. QR Code GeneratorGenerate QR codes for URLs or text using the qrcode and Pillow libraries with a Tkinter interface.Skills Learned: qrcode, Pillow, TkinterDifficulty: IntermediateLearn more about QR Code Generator.17. PDF MergerCreate an app to merge multiple PDF files into one using PyPDF2 and Tkinter for a user-friendly interface.Skills Learned: PyPDF2, Tkinter, file handlingDifficulty: IntermediateLearn more about PDF Merger.18. URL ShortenerBuild a tool to shorten URLs using pyshorteners and PyQt5, with clipboard functionality.Skills Learned: pyshorteners, PyQt5, APIsDifficulty: IntermediateLearn more about URL Shortener.19. Blackjack GameDevelop a Blackjack card game with a GUI, implementing game logic and player decisions.Skills Learned: OOP, PyQt5, random moduleDifficulty: IntermediateLearn more about Blackjack Game.20. Pac-Man GameCreate a simplified Pac-Man game using Pygame, with maze navigation and ghost AI.Skills Learned: Pygame, collision detection, game designDifficulty: IntermediateLearn more about Pac-Man Game.21. Tetris GameBuild a Tetris game with Pygame, featuring falling blocks and scoring mechanics.Skills Learned: Pygame, game logic, event handlingDifficulty: IntermediateLearn more about Tetris Game.22. To-Do List AppCreate a task management app with Tkinter, allowing users to add, edit, and delete tasks.Skills Learned: Tkinter, file handling, listsDifficulty: BeginnerLearn more about To-Do List App.23. Music PlayerDevelop a GUI-based music player with play, pause, and skip controls using Pygame.Skills Learned: Pygame, file handling, GUIDifficulty: IntermediateLearn more about Music Player.24. Currency ConverterBuild a tool to convert currencies using real-time exchange rates via an API.Skills Learned: APIs, requests module, GUIDifficulty: IntermediateLearn more about Currency Converter.25. Text EditorCreate a simple text editor with features like font size adjustment and word count using Tkinter.Skills Learned: Tkinter, file handling, GUIDifficulty: IntermediateLearn more about Text Editor.26. Web ScraperExtract data like article titles from websites using BeautifulSoup and store it in a file.Skills Learned: BeautifulSoup, requests, file handlingDifficulty: IntermediateLearn more about Web Scraper.27. Twitter BotAutomate tweets or retweets using Tweepy and the Twitter API for a specific community.Skills Learned: Tweepy, APIs, automationDifficulty: IntermediateLearn more about Twitter Bot.28. Reddit BotCreate a bot to post or comment on Reddit using PRAW, automating subreddit interactions.Skills Learned: PRAW, APIs, automationDifficulty: IntermediateLearn more about Reddit Bot.29. Discord BotBuild a bot for Discord to automate tasks like posting updates using the Discord API.Skills Learned: Discord API, automation, async programmingDifficulty: IntermediateLearn more about Discord Bot.30. Alarm ClockCreate an alarm clock with customizable times and sound alerts using Tkinter.Skills Learned: Tkinter, time module, audio handlingDifficulty: BeginnerLearn more about Alarm Clock.31. File OrganizerAutomate sorting files into folders based on their extensions using the os module.Skills Learned: os module, file handlingDifficulty: BeginnerLearn more about File Organizer.32. Pomodoro TimerBuild a productivity timer with work and break intervals using Tkinter.Skills Learned: Tkinter, time moduleDifficulty: BeginnerLearn more about Pomodoro Timer.33. Speech to TextConvert spoken words to text using the SpeechRecognition library and save the output.Skills Learned: SpeechRecognition, file handlingDifficulty: IntermediateLearn more about Speech to Text.34. Text to SpeechConvert text to spoken audio using gTTS for audiobooks or announcements.Skills Learned: gTTS, file handlingDifficulty: IntermediateLearn more about Text to Speech.35. Image to SoundGenerate sound based on pixel values in an image using Python libraries.Skills Learned: Image processing, audio generationDifficulty: AdvancedLearn more about Image to Sound.36. Hand Gesture Brightness ControlUse OpenCV and Mediapipe to adjust screen brightness based on hand gestures.Skills Learned: OpenCV, Mediapipe, computer visionDifficulty: AdvancedLearn more about Hand Gesture Control.37. Finger CounterCount raised fingers using OpenCV and Mediapipe for gesture recognition.Skills Learned: OpenCV, Mediapipe, computer visionDifficulty: IntermediateLearn more about Finger Counter.38. Webcam PaintDraw on the screen using a colored marker and webcam with OpenCV.Skills Learned: OpenCV, computer vision, HSV color spaceDifficulty: IntermediateLearn more about Webcam Paint.39. Image EditorCreate a mini-Photoshop with features like cropping and filters using Pillow.Skills Learned: Pillow, image processing, GUIDifficulty: IntermediateLearn more about Image Editor.40. Tower Defense GameBuild a tower defense game with Pygame, including enemy spawning and shooting logic.Skills Learned: Pygame, OOP, game designDifficulty: IntermediateLearn more about Tower Defense Game.41. Connect Four GameCreate a Connect Four game with Pygame, implementing win detection and player turns.Skills Learned: Pygame, numpy, game logicDifficulty: IntermediateLearn more about Connect Four Game.42. Minesweeper GameRecreate Minesweeper in the console or with Pygame, focusing on grid-based logic.Skills Learned: Pygame, grid algorithms, recursionDifficulty: IntermediateLearn more about Minesweeper Game.43. Chess GameBuild a chess game with basic rules and a GUI, allowing two players to compete.Skills Learned: Pygame, game logic, OOPDifficulty: AdvancedLearn more about Chess Game.44. Blog SystemCreate a console-based blog system to add, edit, and delete posts.Skills Learned: File handling, dictionaries, CRUD operationsDifficulty: BeginnerLearn more about Blog System.45. Flask BlogBuild a web-based blog using Flask with post creation and user authentication.Skills Learned: Flask, HTML/CSS, SQLiteDifficulty: IntermediateLearn more about Flask Blog.46. Django E-commerce WebsiteCreate an e-commerce site with Django, including product listings and a cart.Skills Learned: Django, databases, HTML/CSSDifficulty: AdvancedLearn more about Django E-commerce.47. Quiz AppDevelop an interactive quiz app with multiple-choice questions and score tracking.Skills Learned: Lists, dictionaries, conditionalsDifficulty: BeginnerLearn more about Quiz App.48. Horoscope GeneratorCreate a program that generates daily horoscopes based on user star sign input.Skills Learned: Random module, string formattingDifficulty: BeginnerLearn more about Horoscope Generator.49. Budget TrackerTrack expenses and income with a console or GUI-based budget app.Skills Learned: OOP, file handling, TkinterDifficulty: IntermediateLearn more about Budget Tracker.50. Password ManagerBuild a secure password manager to store and retrieve passwords.Skills Learned: Encryption, file handling, TkinterDifficulty: IntermediateLearn more about Password Manager. Create a contact management system to add, edit, and delete contacts.Skills Learned: Dictionaries, file handlingDifficulty: BeginnerLearn more about Contact List.52. Monty Hall SimulationSimulate the Monty Hall problem to explore probability with random choices.Skills Learned: Random module, probabilityDifficulty: IntermediateLearn more about Monty Hall Simulation.53. GIF CreatorGenerate GIFs from images using Pillow for social media or messaging.Skills Learned: Pillow, image processingDifficulty: IntermediateLearn more about GIF Creator.54. Student Management SystemTrack student data like grades and attendance in a console or GUI app.Skills Learned: Databases, Tkinter, CRUD operationsDifficulty: IntermediateLearn more about Student Management System.55. Bank Management SystemCreate a system for deposits, withdrawals, and loans with a GUI interface.Skills Learned: OOP, Tkinter, databasesDifficulty: IntermediateLearn more about Bank Management System.56. Movie Recommendation SystemRecommend movies based on user preferences using basic algorithms.Skills Learned: Data structures, recommendation algorithmsDifficulty: IntermediateLearn more about Movie Recommendation.57. Website BlockerBlock specific websites with a password-protected interface using Python.Skills Learned: File handling, urllib2, GUIDifficulty: IntermediateLearn more about Website Blocker.58. Image DownloaderDownload images from URLs or websites using requests and BeautifulSoup.Skills Learned: Requests, BeautifulSoup, file handlingDifficulty: IntermediateLearn more about Image Downloader.59. Spam BotAutomate keyboard and mouse actions to send repetitive messages using PyAutoGUI.Skills Learned: PyAutoGUI, automationDifficulty: IntermediateLearn more about Spam Bot.60. COVID Vaccine TrackerTrack vaccine availability or stats using APIs and display in a GUI.Skills Learned: APIs, Tkinter, data visualizationDifficulty: IntermediateLearn more about COVID Vaccine Tracker.61. AI Chatbot with StreamlitBuild a chatbot with a web interface using Streamlit for user interaction.Skills Learned: Streamlit, NLP, web developmentDifficulty: IntermediateLearn more about AI Chatbot.62. Food Ordering AppCreate an app for viewing menus and placing orders using dictionaries.Skills Learned: Dictionaries, loops, GUIDifficulty: IntermediateLearn more about Food Ordering App.63. Hacker News Post AnalysisAnalyze post popularity on Hacker News using string manipulation and data handling.Skills Learned: Data analysis, pandas, date handlingDifficulty: IntermediateLearn more about Hacker News Analysis.64. Heart Disease PredictionPredict heart disease risk using a WHO dataset and machine learning.Skills Learned: Pandas, scikit-learn, machine learningDifficulty: AdvancedLearn more about Heart Disease Prediction.65. Movie Ratings AnalysisAnalyze movie ratings for bias using pandas and data visualization.Skills Learned: Pandas, matplotlib, data analysisDifficulty: IntermediateLearn more about Movie Ratings Analysis.66. Traffic Analysis on I-94Explore traffic patterns using pandas and Jupyter Notebook visualizations.Skills Learned: Pandas, matplotlib, Jupyter NotebookDifficulty: IntermediateLearn more about Traffic Analysis.67. Credit Card Customer SegmentationSegment customers using K-means clustering for a credit card company.Skills Learned: Scikit-learn, clustering, pandasDifficulty: AdvancedLearn more about Customer Segmentation.68. eBay Car Sales AnalysisAnalyze used car data from eBay using pandas for insights.Skills Learned: Pandas, data cleaning, visualizationDifficulty: IntermediateLearn more about eBay Car Sales Analysis.69. Music Genre PredictionPredict music genres using machine learning on an audio dataset.Skills Learned: Machine learning, feature engineeringDifficulty: AdvancedLearn more about Music Genre Prediction.70. Uber Pickup AnalysisVisualize Uber ride data on a map using geospatial plotting.Skills Learned: Pandas, geospatial plottingDifficulty: IntermediateLearn more about Uber Pickup Analysis.71. MNIST Digit RecognitionRecognize handwritten digits using a neural network and the MNIST dataset.Skills Learned: TensorFlow, deep learningDifficulty: AdvancedLearn more about MNIST Digit Recognition.72. Face DetectionDetect faces in images using OpenCV's pre-trained models.Skills Learned: OpenCV, computer visionDifficulty: IntermediateLearn more about Face Detection.73. Bee Species DetectionIdentify bee species in images using machine learning techniques.Skills Learned: Machine learning, image processingDifficulty: AdvancedLearn more about Bee Species Detection.74. Hotel Booking Cancellation AnalysisPredict hotel booking cancellations using machine learning on historical data.Skills Learned: Machine learning, pandasDifficulty: AdvancedLearn more about Hotel Booking Analysis.75. Bike Sharing AnalysisAnalyze bike ride data for a sharing startup using pandas.Skills Learned: Pandas, data visualizationDifficulty: IntermediateLearn more about Bike Sharing Analysis.76. Note-Taking App with SpeechCreate a voice-powered note-taking app using SpeechRecognition and Notion API.Skills Learned: SpeechRecognition, APIs, GUIDifficulty: IntermediateLearn more about Note-Taking App.77. Bubble Sort VisualizerVisualize the bubble sort algorithm using Pygame for educational purposes.Skills Learned: Pygame, algorithms, visualizationDifficulty: IntermediateLearn more about Bubble Sort Visualizer.78. Search Engine APIBuild a simple search engine backend using a Python framework like Flask.Skills Learned: Flask, APIs, backend developmentDifficulty: AdvancedLearn more about Search Engine API.79. Desktop NotifierCreate an app to send desktop notifications for reminders or alerts.Skills Learned: Plyer, time module, GUIDifficulty: BeginnerLearn more about Desktop Notifier.80. Torrent SaverSave torrent files to Google Drive using Python automation.Skills Learned: Google Drive API, file handlingDifficulty: IntermediateLearn more about Torrent Saver.81. Audiobook PlayerBuild a player for audiobooks with speed control using Pygame.Skills Learned: Pygame, audio handling, GUIDifficulty: IntermediateLearn more about Audiobook Player.82. Command Line ChatCreate a simple chat app for the command line using sockets.Skills Learned: Sockets, networking, threadingDifficulty: IntermediateLearn more about Command Line Chat.83. Telegram BotDevelop a Telegram bot for community tasks using the Telegram API.Skills Learned: Telegram API, automationDifficulty: IntermediateLearn more about Telegram Bot.84. Email AutomatorAutomate sending emails like reminders using smtplib.Skills Learned: smtplib, string formattingDifficulty: IntermediateLearn more about Email Automator.85. WhatsApp AutomatorAutomate WhatsApp messages for birthdays or tasks using PyWhatKit.Skills Learned: PyWhatKit, automationDifficulty: IntermediateLearn more about WhatsApp Automator.86. Eulerian Path FinderFind Eulerian paths or cycles in a graph, exploring graph theory.Skills Learned: Graph algorithms, OOPDifficulty: AdvancedLearn more about Eulerian Path Finder.87. Fake News DetectorDetect fake news using machine learning and text analysis.Skills Learned: NLP, scikit-learn, pandasDifficulty: AdvancedLearn more about Fake News Detector.88. Simple BlockchainImplement a basic blockchain with transaction tracking and tamper detection.Skills Learned: Cryptography, data structuresDifficulty: AdvancedLearn more about Simple Blockchain.89. Bus Reservation SystemCreate a system to reserve bus seats with a GUI interface.Skills Learned: Tkinter, databases, CRUD operationsDifficulty: IntermediateLearn more about Bus Reservation System.90. Color DetectorDetect colors in an image and output their hex codes using OpenCV.Skills Learned: OpenCV, image processingDifficulty: IntermediateLearn more about Color Detector.91. Attendance TrackerAutomate attendance tracking from an Excel sheet with a GUI.Skills Learned: Pandas, Tkinter, file handlingDifficulty: IntermediateLearn more about Attendance Tracker.92. Fibonacci CheckerCheck if a number belongs to the Fibonacci sequence using recursion.Skills Learned: Recursion, memoizationDifficulty: IntermediateLearn more about Fibonacci Checker.93. Voice AssistantCreate a voice assistant for tasks like web searches or reminders.Skills Learned: SpeechRecognition, APIs, automationDifficulty: AdvancedLearn more about Voice Assistant.94. Markov Chain Lyrics GeneratorGenerate song lyrics using a Markov chain model for AI creativity.Skills Learned: Markov chains, NLP, data structuresDifficulty: AdvancedLearn more about Markov Chain Lyrics.95. Binary Search ImplementationImplement a binary search algorithm and visualize it with Pygame.Skills Learned: Algorithms, Pygame, visualizationDifficulty: IntermediateLearn more about Binary Search.96. File RenamerAutomate renaming files in a folder based on user-defined conditions.Skills Learned: os module, file handlingDifficulty: BeginnerLearn more about File Renamer.97. Instagram Photo DownloaderDownload photos from Instagram using Python libraries.Skills Learned: APIs, requests, file handlingDifficulty: IntermediateLearn more about Instagram Photo Downloader.98. Speech ExtractionExtract text from audio recordings using SpeechRecognition.Skills Learned: SpeechRecognition, file handlingDifficulty: IntermediateLearn more about Speech Extraction.99. World Cup PredictorPredict World Cup outcomes using Poisson distribution and data analysis.Skills Learned: Pandas, probability, data analysisDifficulty: AdvancedLearn more about World Cup Predictor.100. Memory GameCreate a memory game with Pygame to test matching skills with cards.Skills Learned: Pygame, nested loops, game logicDifficulty: IntermediateLearn more about Memory Game. Once you have built a few fun python projects, it is time to showcase your work and get valuable feedback from the coding community. Sharing your fun python projects not only helps build your portfolio but also connects you with other learners and professionals who can inspire and guide you.Here are some great platforms to share your fun python projects :GitHub : Ideal for version control and displaying your fun python projects in a professional portfolioReplit : Perfect for collaborative coding and receiving feedback on your fun python projects in real timeKaggle : A great space for data-driven fun python projects and competitionsDev.to : Share blog posts, tutorials, or insights about your fun python projects and coding journeyPublishing your fun python projects helps demonstrate initiative, creativity, and technical skill. These are qualities that stand out to colleges, internships, and future employers. If you are looking for a way to make coding more enjoyable and less intimidating, fun python projects are an excellent place to begin. These projects let you apply what you have learned in a creative and meaningful way, whether you are building a simple calculator, a text-based adventure game, or a quirky chatbot. Each fun python project helps you strengthen your problem-solving abilities, reinforce core programming concepts like loops and conditionals, and develop confidence with real tools used in the industry. They make coding feel like play, not just practice. Start by choosing a fun python project that aligns with your interests, maybe something related to your favorite hobby, a personal challenge you want to solve, or a tool that helps others. Do not be afraid to take risks, try new libraries, or even break things. The hands-on experience you gain from working on fun python projects will help you build a strong portfolio, stand out in college applications or tech internships, and spark a lifelong love for computer science. Ready for more inspiration? Let me know, and I will share over 50 exciting fun python project ideas complete with source code to get you started. AI Scholars Live Online is a 10-session (25-hour) program that exposes high school students to fundamental AI concepts and guides them to build a socially impactful project. Taught by our team of graduate students from Stanford, MIT, and more, students receive a personalized learning experience in small groups with a student-teacher ratio of 5:1. Kuda Cee DSA to Development: A Complete GuideBeginner to AdvanceJAVA Backend Development - LiveIntermediate and AdvanceFull Stack Development with React & Node JS - Project Based TrainingBeginner to AdvanceTech Interview 101 - From DSA to System Design for Working ProfessionalsBeginner to AdvanceC++ Programming Course Online - Complete Beginner to AdvancedBeginner to AdvanceJava Programming Online Course [Complete Beginner to Advanced]Beginner to AdvancePage 2Our website uses cookiesWe use cookies to ensure you have the best browsing experience on our website. By using our site, you acknowledge that you have read and understood our Cookie Policy & Privacy Policy Disclosure: Hackr.io is supported by its audience. When you purchase through links on our site, we may earn an affiliate commission. Share — copy and redistribute the material in any medium or format for any purpose, even commercially. Adapt — remix, transform, and build upon the material for any purpose, even commercially. The licensor cannot revoke these freedoms as long as you follow the license terms. Attribution — You must give appropriate credit , provide a link to the license, and indicate if changes were made . You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use. ShareAlike — If you remix, transform, or build upon the material, you must distribute your contributions under the same license as the original. No additional restrictions — You may not apply legal terms or technological measures that legally restrict others from doing anything the license permits. You do not have to comply with the license for elements of the material in the public domain or where your use is permitted by an applicable exception or limitation . No warranties are given. The license may not give you all of the permissions necessary for your intended use. For example, other rights such as publicity, privacy, or moral rights may limit how you use the material. Python Projects - Beginner to Advanced Number guessing game in Python 3 and C Python Program for word Guessing Game Hangman Game in Python 21 Number game in Python Mastermind Game using Python 2048 Game in Python Python | Program to implement simple FLAMES game Python | PokÃ©mon Training Game Python program to implement Rock Paper Scissor game Taking Screenshots using pyscreenshot in Python Desktop Notifier in Python Get Live Weather Desktop Notifications Using Python How to use pynput to make a Keylogger? Python - Cows and Bulls game Simple Attendance Tracker using Python Higher-Lower Game with Python Fun Fact Generator Web App in Python Check if two PDF documents are identical with Python Creating payment receipts using Python How To Create a Countdown Timer Using Python? Convert emoji into text in Python Create a Voice Recorder using Python Create a Screen recorder using Python How to Build a Simple Auto-Login Bot with Python How to make a Twitter Bot in Python? Building WhatsApp bot on Python Create a Telegram Bot using Python Twitter Sentiment Analysis using Python Employee Management System using Python How to make a Python auto clicker? Instagram Bot using Python and InstaPy File Sharing App using Python Send message to Telegram user using Python Python | Whatsapp birthday bot Corona HelpBot Amazon product availability checker using Python Python | Fetch your gmail emails from a particular user How to Create a Chatbot in Android with BrainShop API? Spam bot using PyAutoGUI Hotel Management System Build a COVID19 Vaccine Tracker Using Python Email Id Extractor Project from sites in Scrapy Python Automating Scrolling using Python-Opencv by Color Detection How to scrape data from google maps using Python ? Scraping weather data using Python to get umbrella reminder on email Scraping Reddit using Python How to fetch data from Jira in Python? Scrape most reviewed news and tweet using Python Extraction of Tweets using Tweepy Predicting Air Quality Index using Python Scrape Content from Dynamic Websites Automate Instagram Messages using Python Python | Automating Happy Birthday post on Facebook using Selenium Automatic Birthday mail sending with Python Automated software testing with Python Python | Automate Google Search using Selenium Automate linkedin connections using Python Automated Trading using Python Automate the Conversion from Python2 to Python3 Bulk Posting on Facebook Pages using Selenium Share WhatsApp Web without Scanning QR code using Python Automate WhatsApp Messages With Python using Pywhatkit module How to Send Automated Email Messages in Python Automate backup with Python Script Automated software testing with Python Hotword detection with Python Automate linkedin connections using Python Create First GUI Application using Python-Tkinter Simple GUI calculator using Tkinter-Python Python - Compound Interest GUI Calculator using Tkinter Python | Loan calculator using Tkinter Rank Based Percentile Gui Calculator using Tkinter Standard GUI Unit Converter using Tkinter in Python Create Table Using Tkinter Python | GUI Calendar using Tkinter File Explorer in Python using Tkinter Python | ToDo GUI Application using Tkinter Python: Weight Conversion GUI using Tkinter Python: Age Calculator using Tkinter Python | Create a GUI Marksheet using Tkinter Python | Create a digital clock using Tkinter Create Countdown Timer using Python-Tkinter Tkinter Application to Switch Between Different Page Frames Color game using Tkinter in Python Python | Simple FLAMES game using Tkinter Simple registration form using Python Tkinter Image Viewer App in Python using Tkinter How to create a COVID19 Data Representation GUI? GUI to Shutdown, Restart and Logout from the PC using Python Create a GUI to extract Lyrics from song Using Python Application to get live USD/INR rate Using Python Build an Application for Screen Rotation Using Python Build an Application to Search Installed Application using Python Text detection using Python Python - Spell Corrector GUI using Tkinter Make Notepad using Tkinter Sentiment Detector GUI using Tkinter - Python Create a GUI for Weather Forecast using openweathermap API in Python Build a Voice Recorder GUI using Python Create a Sideshow application in Python Visiting Card Scanner GUI Application using Python Create digital clock using Python-Turtle Draw a Tic Tac Toe Board using Python-Turtle Draw Chess Board Using Turtle in Python Draw an Olympic Symbol in Python using Turtle Draw Rainbow using Turtle Graphics in Python How to Make an Indian Flag in Python Draw moving object using Turtle in Python Create a simple Animation using Turtle in Python Create a Simple Two Player Game using Turtle in Python Flipping Tiles (memory game) using Python3 Create pong game using Python - Turtle Python | Program to extract frames using OpenCV Displaying the coordinates of the points clicked on the image using Python-OpenCV White and black dot detection using OpenCV | Python Python | OpenCV BGR color palette with trackbars Draw a rectangular shape and extract objects using Python's OpenCV Drawing with Mouse on Images using Python-OpenCV Text Detection and Extraction using OpenCV and OCR Invisible Cloak using OpenCV | Python Project Background subtraction - OpenCV ML | Unsupervised Face Clustering Pipeline Pedestrian Detection using OpenCV-Python Saving Operated Video from a webcam using OpenCV Face Detection using Python and OpenCV with webcam Gun Detection using Python-OpenCV Multiple Color Detection in Real-Time using Python-OpenCV Detecting objects of similar color in Python using OpenCV Opening multiple color windows to capture using OpenCV in Python Python | Play a video in reverse mode using OpenCV Template matching using OpenCV in Python Cartooning an Image using OpenCV - Python Vehicle detection using OpenCV Python Count number of Faces using Python - OpenCV Live Webcam Drawing using OpenCV Detect and Recognize Car License Plate from a video in real time Track objects with Camshift using OpenCV Replace Green Screen using OpenCV- Python Python - Eye blink detection project Connect your android phone camera to OpenCV - Python Determine The Face Tilt Using OpenCV - Python Right and Left Hand Detection Using Python Brightness Control With Hand Detection using OpenCV in Python Creating a Finger Counter Using Computer Vision and OpenCv in Python Python Web Development With Django How to Create an App in Django ? Weather app using Django - Python Django Sign Up and login with confirmation Email | Python ToDo webapp using Django How to Send Email with Django Django project to create a Comments System Voting System Project Using Django Framework Determine The Face Tilt Using OpenCV - Python How to add Google reCAPTCHA to Django forms ? E-commerce Website using Django College Management System using Django - Python Project Create Word Counter App using Django Speak the meaning of the word using Python Convert PDF File Text to Audio Speech using Python Speech Recognition in Python using Google Speech API Convert Text to Speech in Python Python Text To Speech | pyttsx module Python: Convert Speech to text and text to Speech Personal Voice Assistant in Python Build a Virtual Assistant Using Python Python | Create a simple assistant using Wolfram Alpha API. Voice Assistant using python Voice search Wikipedia using Python Language Translator Using Google API in Python How to make a voice assistant for E-mail in Python? Voice Assistant for Movies using Python So you're done with your first Python programming course? Congrats! Now you're probably wondering: How can I practice my Python skills now? What Python projects for beginners can I build next? Now that you know the basics of Python programming, it's time to put your skills to good use and build something useful. This is when the fun part of learning Python starts! But where can you find Python project ideas that are easy and fun to build? The truth is: the sooner you start building your first Python projects, the better. You should start practicing on fun and easy Python projects asap. This will help you understand how the language works much better as you create something on your own, step by step. Building your own Python projects is the only way to put together a stellar portfolio, which is your #1 goal if you want to become a Python developer and start a career in tech. It doesn't matter how big or small your projects are, as long as you're building them without following a video tutorial or a book every step of the way. You need to work independently as much as you can to learn how to apply your skills on your own. To help you get started, I'll walk you through 10 fun Python projects for beginners in this post. All of them are easy, fun, and perfect for you if you are relatively new to programming. By the end of this article, you will be armed with a bunch of Python projects for practice to take your skills to the next level starting right now! Here are a few related articles about Python programming: If you are not familiar, Python is a high-level, object-oriented programming language. Python is a multi-purpose programming language, so it's very versatile. Companies use Python for a plethora of projects in areas like: Data collection and database creation Data analysis and science Machine learning algorithms Artificial intelligence Web development Scripting and automation Hence, if you are new to coding and not sure what area you wish to specialize in, Python is a tool you can use for a plethora of different projects in the future. Python is relatively easy to learn, too. Its syntax is clean and clear, and it reads a lot like English. If you are new to programming, you will spend less time troubleshooting your code and more time learning and building projects. These are just a few reasons why Python is the perfect first programming language to learn. Because of its advantages, Python has become one of the most popular coding languages worldwide; Python currently ranks at #1 on the TIOBE Index, for example: Also, over 48% of respondents in the 2022 Stack Overflow Survey had used it. Because Python is so popular, it's also in high demand on the job market. On average, Junior Python Developers earn $96,697 annually according to Indeed. I've written more about my own Python journey in my guide on how I learned Python in 4 weeks (with no experience). Now that you know what Python is, let's look at some interesting Python projects you can build as a beginner. Once you have learned the basics of Python, the most important thing is to start using it for solving small problems asap. If you have finished your first Python tutorial or online course, you're a more capable programmer than you may think. You have a basic sense of how Python works and how you can model real-life situations in small programs. I encourage you to keep a running list of small Python project ideas of your own in Google Sheets or Trello, for example. Think of simple web apps you would want to create, small data sets you could analyze, or basic 2D video games you would like to program, for instance. If you're short of ideas right now, let's go through a few project ideas for inspiration! Here are 10 simple Python projects for beginners to build: Odd or Even? Guessing the Number Horoscope Rock Paper Scissors Hangman Acronym Password Generator Word Count Pong Game Tic Tac Toe Make sure you share this post with others and pin it for later! Let's start with a super easy project you can build right away. Write a small Python program that asks the user for a number and tells them if it is odd or even. To help you get started, here's a good article about the Modulo operator in Python. Making a user guess a number is another easy Python program you can write in just a few lines of code. It should generate a random number between, let's say, 0 and 100. The player then needs to guess what that number is. Should the user guess wrong, the program should respond by telling them their guess is either too low or too high. When the user guesses right, your program should ask them if they want to play again. For a little added challenge, you can limit the number of guesses to 5, for example. Create a simple horoscope program that asks the user for their star sign and outputs a fun horoscope for them. Bear in mind that your program should display an error message if the user types in their sign wrong. Rock Paper Scissors (or RPS) is one of my favorite Python projects for beginners. It's simple enough but still challenging since you need to think about how to turn the game rules into logic for the program. The rules are simple: Paper wins over rock Rock wins over scissors Scissors win over paper Have your program generate a random answer for the computer – but don't display it. Then, ask the player for their answer. Hangman is one of the best Python projects for beginners who want a bit of a challenge. The game is about guessing a random word with a limited number of guesses. Although Hangman may sound rather simple, you need to consider a few key points, such as: You need to set a maximum limit for guesses. The player needs to be notified about the remaining number of guesses. Your player needs to be able to input their guesses. To get started, you need a way to generate random words to be guessed. The best way is to simply create a separate text file where you store the words or short phrases. This Python project will probably take you a bit longer, but it's great practice. You will have to think about random choice, variables, boolean values, inputs and outputs, strings, length, and much more. To build a very basic text-based Hangman game, here's a step-by-step tutorial video for some helpful tips: An acronym Python program will take a given phrase or text and convert it into its acronym. That is, a word that consists of the first letters of each word in the text. To make your acronym easier to read, go ahead and turn the letters into uppercase with Python. Also, think about how the program will take text as input from the user while separating that user input from the logic itself. Create a Python program that generates a random password for the user. Make sure your program takes a few inputs from the user: How long should the password be? How many characters should there be? Should it have both uppercase and lowercase letters? Should it include numbers and special symbols, too? The best part about this small Python project is that you can actually use it for generating strong passwords for your own user accounts across the Web! Write a Python program that takes a text file as an input and outputs how many times each word occurs in the text. You can start small by counting the number of words in a list, for example. Use the Counter collection for this. Once you understand how the collection works, move on to more advanced inputs, such as longer text files. You can access complete book scripts for free at Project Gutenberg for this. Creating a simple Pong game is one of the easiest projects to learn how to use Python for simple video games. You can use the Pygame library to speed up your workflow for designing and writing your game. Using Python to build simple but fun games is definitely one of the most enjoyable ways to learn the language. Moreover, you will get more ideas for developing your game further and adding new features while you're playing. If you're ready to start building your first video game with Python right away, here's a full video course from freeCodeCamp including a Pong game tutorial: Tic Tac Toe is a fun Python project to build in a single day. It's the game where two players take turns marking the 3×3 grid with cross and circle symbols. The player who gets three marks in a horizontal, vertical, or diagonal row is the winner. This little Python project is fun to build but it requires you to think about the logic of the game in your code. The players need to take turns and the game needs to keep going until (1) someone wins or (2) the grid is full, for example. Hence, before you start writing actual Python code, map out the different features of the game. First, write down all the steps your program needs to go through. Only then start writing out those steps using Python. If you want to build your Tic Tac Toe game right now, here's a beginner-level walkthrough tutorial from Clever Programmer: Conducting data analysis with Python is a great idea for an intermediate Python project. Python supports a number of powerful libraries you can use for analyzing, visualizing, and mining data. Small data analysis projects will familiarize you with a few popular Python libraries, such as Matplotlib, NumPy, and Pandas, for instance. My first Python book, Python Crash Course by Eric Matthes, had a fun project where I learned how to do data analysis with Python. If you're looking for an all-in-one Python book for absolute beginners, I definitely recommend checking it out. If you learn better by watching videos, here's one of my favorite videos for learning the basics of Matplotlib and Pandas for absolute beginners: If the projects above feel too easy, you may be looking for more advanced Python project ideas. If you are familiar with the fundamental Python libraries and a few external libraries, you should know how to install external libraries and use them for your projects. If that's the level you are at right now, here are 20 advanced Python projects for you: Now that you have a few ideas for practical Python projects for beginners, let's go through a few essentials about how to go about building them. What you want to do first is to make sure you find good resources for learning the basics of how Python works. You can choose books, online courses, or Python YouTube tutorials for this. Choose whatever method fits your learning style the best. Next, you should study the code you see in your tutorials or books. Familiarize yourself with how the syntax works and how it follows a certain logic in each program. This step is super important for understanding how to solve problems with code. Step-by-step, you will start feeling more comfortable with reading and writing Python code yourself. And by this, I mean very simple programs, nothing too advanced. Once you have solved a few small exercises, you're pretty much all set for building small programs on your own. Hence, I strongly encourage you to break away from your tutorials and try to solve a few small problems by yourself using Python. Read next: The Best Way to Learn to Code: Beginner's Guide Are you ready to start building your first Python project? Great! The most important thing to keep in mind is to start small. You don't want to set your expectations too high. Otherwise, you will just end up feeling overwhelmed and disappointed. Now, before you start writing Python code, let me walk you through a few helpful tips first. Essentially, this is what it all boils down to: For any Python program you write, you need to understand the underlying problem and figure out how to solve it with Python step-by-step. If solving problems feels fun to you, programming could just be your thing! So what you want to do when you start a Python project is: Figure out your problem Break it down to smaller steps Map out the individual steps your program should perform Solve each step individually These four steps will help you stay focused on what you're supposed to do. Also, they will keep you motivated as you solve smaller parts of the main problem one by one. The fun Python projects for beginners above should keep you busy building your own programs for a while. Start with the first ones since they are the easiest. Then, challenge yourself to learn Python by practicing with more complex projects. They are still quite easy, but they include more steps to solving the main problem. Therefore, you will need to think more about how to construct your program to perform all the necessary steps. All in all, building Python projects boils down to learning how to solve problems by yourself using Python code. There are tons of project ideas to explore and you will probably come up with a few ideas of your own as you go. I hope you found a few Python project ideas in this post. If you have an idea I should add to the article, drop me a line in the comments below! If you are looking to learn Python for machine learning, check out these machine learning project ideas for Python, too. To learn more about Python, here are a few related posts for you: If you liked this article about Python projects for beginners, please leave me a comment! P.S. If you found my Python project ideas useful, please share this post with others! Thanks for your support! Happy coding!– Mikke Learn how to integrate MediaPipe's Tasks API into a React app for fast, in-browser object detection using your webcam. Emmanuel John Jul 17, 2025 · 10 min read Integrating AI into modern frontend apps can be messy. This tutorial shows how the Vercel AI SDK simplifies it all, with streaming, multimodal input, and generative UI. Ikeh Akinyemi Jul 16, 2025 · 13 min read Interviewing for a software engineering role? Hear from a senior dev leader on what he looks for in candidates, and how to prepare yourself. Andrew Evans Jul 16, 2025 · 12 min read Set up real-time video streaming in Next.js using HLS.js and alternatives, exploring integration, adaptive streaming, and

token-based authentication. Jude Miracle Jul 15, 2025 · 19 min read UX Design We've all made silly mistakes in apps — clicked the wrong thing, deleted a file too soon. This guide walks through 12 real UX examples that help users avoid those moments entirely. Shalitha Suranga Jul 17, 2025 · 11 min read Dev Learn how to integrate MediaPipe's Tasks API into a React app for fast, in-browser object detection using your webcam. Emmanuel John Jul 17, 2025 · 10 min read Product Management AI agrees too easily. That's a problem. Learn how to prompt it to challenge your thinking and improve your product decisions. Raluca Piteiu-Apostol Jul 17, 2025 · 5 min read Dev Integrating AI into modern frontend apps can be messy. This tutorial shows how the Vercel AI SDK simplifies it all, with streaming, multimodal input, and generative UI. Ikeh Akinyemi Jul 16, 2025 · 13 min read Product Management Chrissie Lamond, VP of Product at Mansueto Ventures, talks about how she builds product experiences across audience segments. Marta Randall Jul 16, 2025 · 7 min read UX Design Users rarely welcome change, even when it improves their experience. Learn how to communicate updates, handle pushback, and keep users from jumping ship. Yaroslav Malymon Jul 15, 2025 · 8 min read Dev Set up real-time video streaming in Next.js using HLS.js and alternatives, exploring integration, adaptive streaming, and token-based authentication. Jude Miracle Jul 15, 2025 · 19 min read Product Management AI governance is now a product feature. Learn how to embed trust, transparency, and compliance into your build cycles. Kayode Adeniyi Jul 15, 2025 · 5 min read Product Management Michal Ochnicki talks about the importance of ensuring that the ecommerce side of a business is complementary to the whole organization. Sibel Bagcilar Jul 14, 2025 · 6 min read Dev Not sure if low-code is right for your next project? This guide breaks down when to use it, when to avoid it, and how to make the right call. Popoola Temitope Jul 11, 2025 · 7 min read Dev Compare Firebase Studio, Lovable, and Replit for AI-powered app building. Find the best tool for your project needs. Emmanuel John Jul 11, 2025 · 7 min read Product Management Christina Valls shares how her teams have transformed digital experiences at Cedars-Sinai, including building a digital scheduling platform. Sibel Bagcilar Jul 11, 2025 · 10 min read Python is one of the most versatile programming languages, offering endless possibilities for fun and educational projects. Whether you're a beginner learning the ropes or an experienced coder seeking to refine your skills, Python is an excellent language to use for diverse project ideas. In this blog post, we'll explore over 30 fun Python project ideas, ranging from simple to advanced, covering various domains like web development, game creation, data science, automation, and more. If you're just starting with Python, these beginner projects will help you get a good grasp of the basics like loops, conditionals, functions, and simple algorithms.1. Number Guessing GameBuild a small game where the computer picks a random number and the user has to guess it. Provide hints like "too high" or "too low" based on the guess.Skills learned: User input, random module, conditionals.2. Basic CalculatorCreate a basic calculator that can perform addition, subtraction, multiplication, and division. You can add more operations as you advance.Skills learned: Functions, basic math operations, input handling.3. Rock, Paper, Scissors GameDevelop a simple Rock, Paper, Scissors game where the user plays against the computer. The game should declare a winner after each round.Skills learned: Conditionals, loops, random choices. Code a countdown timer that takes input from the user in seconds and counts down to zero with a message displayed when time is up.Skills learned: Time module, loops, input validation.Create a simple text-based to-do list app where users can add, view, and remove tasks.Skills learned: Lists, file I/O (for saving tasks), loops.Build a Mad Libs game where the program asks the user for a list of words, then inserts those words into a story template.Skills learned: String manipulation, user input.Develop a program that simulates rolling a die by generating a random number between 1 and 6 when prompted.Skills learned: Random module, loops, conditionals.Create a random password generator where users can set the length and characters (e.g., letters, numbers, symbols) they want in the password.Skills learned: String and random module.Once you're comfortable with the basics, it's time to move on to intermediate-level projects that integrate more advanced concepts like object-oriented programming, file handling, and data structures.9. Weather App Using APIUse a weather API (like OpenWeatherMap) to fetch real-time weather data based on the user's input location.Skills learned: API interaction, JSON parsing, requests module.10. Simple Web ScraperBuild a web scraper that can extract data from a webpage, such as headlines from a news website or product prices.Skills learned: BeautifulSoup, requests module, HTML parsing.11. Tic-Tac-Toe GameCode a simple Tic-Tac-Toe game that can be played in the terminal or with a graphical interface using libraries like Tkinter.Skills learned: Game logic, 2D arrays, user input handling.12. Expense TrackerDevelop a personal expense tracker that records income and expenses, and categorizes transactions for budgeting.Skills learned: File I/O, lists, loops, dictionaries.13. Currency ConverterBuild a program that converts an amount from one currency to another using exchange rates from an API.Skills learned: API requests, math operations, conditionals.14. Hangman GameCreate a Hangman game where the computer randomly picks a word, and the user tries to guess the letters before running out of tries.Skills learned: String manipulation, loops, conditionals.15. Flashcard ApplicationDevelop a flashcard app that allows users to create a set of flashcards for learning vocabulary or any other subject.Skills learned: Lists, file handling, input validation.16. Simple Blog WebsiteBuild a simple blog website using Python's Flask framework, where users can post articles, and you can manage them from an admin panel.Skills learned: Web development, Flask, HTML/CSS, database integration (SQLite).17. Quiz ApplicationCreate a quiz app that asks multiple-choice questions, records the user's score, and provides feedback at the end.Skills learned: Lists, dictionaries, conditionals, loops.18. Digital ClockBuild a digital clock using the Tkinter library that displays the current time in hours, minutes, and seconds.Skills learned: Tkinter, time module.19. File OrganizerWrite a program that automatically organizes files in a folder into subfolders based on file extensions.Skills learned: File handling, os module, conditionals.Advanced-Level Python ProjectsAdvanced projects will challenge you to use complex libraries and frameworks, implement algorithms, or work on real-world problems.20. Chatbot Using NLPDevelop a simple chatbot using Natural Language Processing (NLP) that can understand user input and provide appropriate responses.Skills learned: NLP libraries (NLTK, spaCy), AI basics, text processing.21. Sentiment AnalysisCreate a sentiment analysis tool that analyzes the sentiment (positive, negative, neutral) of a given text or dataset (e.g., tweets, reviews).Skills learned: Machine learning, text classification, libraries like Scikit-learn.22. Stock Price PredictorBuild a stock price prediction model using historical data and machine learning algorithms like linear regression or LSTM (Long Short-Term Memory).Skills learned: Pandas, NumPy, Scikit-learn, machine learning algorithms.23. Face Recognition SystemDevelop a face recognition system using OpenCV and a pre-trained machine learning model.Skills learned: OpenCV, machine learning, image processing.24. Django E-commerce WebsiteCreate a full-fledged e-commerce website using Django with features like user authentication, product listings, cart functionality, and payment integration.Skills learned: Django framework, HTML/CSS, databases (PostgreSQL/MySQL).25. Automated Email SenderWrite a program that automates sending emails, such as reminders or newsletters, using the smtplib module in Python.Skills learned: smtplib, email protocols, string formatting.26. File Encryption ProgramCreate a program that encrypts and decrypts files using a secret key for secure file sharing.Skills learned: Cryptography, file handling, security protocols.27. Real-Time Object DetectionDevelop a real-time object detection app using a machine learning model like YOLO (You Only Look Once) to recognize objects via a webcam feed.Skills learned: OpenCV, TensorFlow, computer vision algorithms.28. Virtual AssistantBuild your own virtual assistant that can perform tasks like searching the web, setting reminders, or sending emails using voice commands.Skills learned: Speech recognition, web scraping, task automation.29. Blockchain ImplementationImplement a simple blockchain from scratch to understand how blockchain technology works, including features like transaction validation and consensus.Skills learned: Cryptography, distributed systems, data structures.30. Automated Resume ParserCreate a tool that parses resumes and extracts key information like name, skills, and experience, which can be useful for HR departments.Skills learned: NLP, regex, file handling.Bonus Projects31. Sudoku SolverWrite a program that solves Sudoku puzzles using backtracking algorithms.Skills learned: Algorithms, recursion, problem-solving.32. Instagram BotDevelop a bot that automatically likes and comments on Instagram posts based on specific hashtags.Skills learned: Web scraping, Instagram API, automation.33. Interactive DictionaryBuild an interactive dictionary that allows users to search for words and provides definitions, synonyms, and antonyms using a dictionary API.Skills learned: API usage, file handling, dictionaries.34. Expense Manager with Graphical InterfaceCreate an expense manager with a graphical interface using Tkinter, allowing users to track and visualize their spending habits.Skills learned: Tkinter, file handling, matplotlib. These 30+ fun Python project ideas offer a wide variety of learning experiences and challenges, ranging from simple games to more complex machine learning and web development projects. Whether you're a beginner or an advanced coder, these projects will help you solidify your Python skills and expand your programming capabilities.So, pick a project, dive into the code, and start building something fun and educational today!About Inspirit AIAI Scholars Live Online is a 10-session (25-hour) program that exposes high school students to fundamental AI concepts and guides them to build a socially impactful project. Taught by our team of graduate students from Stanford, MIT, and more, students receive a personalized learning experience in small groups with a student-teacher ratio of 5:1. Naman Agarwal Programming paradigm based on the concept of objects "Object-oriented" redirects here. For other meanings of object-oriented, see Object-orientation. UML notation for a class. This Button class has variables for data, and functions. Through inheritance, a subclass can be created as a subset of the Button class. Objects are instances of a class. Object-oriented programming (OOP) is a programming paradigm based on the concept of objects.[1] Objects can contain data (called fields, attributes or properties) and have actions they can perform (called procedures or methods and implemented in code). In OOP, computer programs are designed by making them out of objects that interact with one another.[2][3] Many of the most widely used programming languages (such as C++, Java,[4] and Python) support object-oriented programming to a greater or lesser degree, typically as part of multiple paradigms in combination with others such as imperative programming and declarative programming. Significant object-oriented languages include Ada, ActionScript, C++, Common Lisp, C#, Dart, Eiffel, Fortran 2003, Haxe, Java,[4] JavaScript, Kotlin, Logo, MATLAB, Objective-C, Object Pascal, Perl, PHP, Python, R, Raku, Ruby, Scala, SIMSCRIPT, Simula, Smalltalk, Swift, Vala and Visual Basic.NET. The idea of "objects" in programming started with the artificial intelligence group at MIT in the late 1950s and early 1960s. Here, "object" referred to LISP atoms with identified properties (attributes).[5][6] Another early example was Sketchpad created by Ivan Sutherland at MIT in 1960–1961. In the glossary of his technical report, Sutherland defined terms like "object" and "instance" (with the class concept covered by "master" or "definition"), albeit specialized to graphical interaction.[7] Later, in 1968, AED-0, MIT's version of the ALGOL programming language, connected data structures ("plexes") and procedures, prefiguring what were later termed "messages", "methods", and "member functions".[8][9] Topics such as data abstraction and modular programming were common points of discussion at this time. Meanwhile, in Norway, Simula was developed during the years 1961–1967.[8] Simula introduced essential object-oriented ideas, such as classes, inheritance, and dynamic binding.[10] Simula was used mainly by researchers involved with physical modelling, like the movement of ships and their content through cargo ports.[10] Simula is generally accepted as being the first language with the primary features and framework of an object-oriented language.[11] I thought of objects being like biological cells and/or individual computers on a network, only able to communicate with messages (so messaging came at the very beginning – it took a while to see how to do messaging in a programming language efficiently enough to be useful). Alan Kay, [1] Influenced by both MIT and Simula, Alan Kay began developing his own ideas in November 1966. He would go on to create Smalltalk, an influential object-oriented programming language. By 1967, Kay was already using the term "object-oriented programming" in conversation.[1] Although sometimes called the "father" of object-oriented programming,[12] Kay has said his ideas differ from how object-oriented programming is commonly understood, and has implied that the computer science establishment did not adopt his notion.[1] A 1976 MIT memo co-authored by Barbara Liskov lists Simula 67, CLU, and Alphard as object-oriented languages, but does not mention Smalltalk.[13] In the 1970s, the first version of the Smalltalk programming language was developed at Xerox PARC by Alan Kay, Dan Ingalls and Adele Goldberg. Smalltalk-72 was notable for use of objects at the language level and its graphical development environment.[14] Smalltalk was a fully dynamic system, allowing users to create and modify classes as they worked.[15] Much of the theory of OOP was developed in the context of Smalltalk, for example multiple inheritance.[16] In the late 1970s and 1980s, object-oriented programming rose to prominence. The Flavors object-oriented Lisp was developed starting 1979, introducing multiple inheritance and mixins.[17] In August 1981, Byte Magazine highlighted Smalltalk and OOP, introducing these ideas to a wide audience.[18] LOOPS, the object system for Interlisp-D, was influenced by Smalltalk and Flavors, and a paper about it was published in 1982.[19] In 1986, the first Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA) was attended by 1,000 people. This conference marked the beginning of efforts to consolidate Lisp object systems, eventually resulting in the Common Lisp Object System. In the 1980s, there were a few attempts to design processor architectures that included hardware support for objects in memory, but these were not successful. Examples include the Intel iAPX 432 and the Linn Smart Rekursiv. In the mid-1980s, new object-oriented languages like Objective-C, C++, and Eiffel emerged. Objective-C was developed by Brad Cox, who had used Smalltalk at ITT Inc. Bjarne Stroustrup created C++ based on his experience using Simula for his PhD thesis.[14] Bertrand Meyer produced the first design of the Eiffel language in 1985, which focused on software quality using a design by contract approach.[20] In the 1990s, object-oriented programming became the main way of programming, especially as more languages supported it. These included Visual FoxPro 3.0,[21][22] C++,[23] and Delphi[citation needed]. OOP became even more popular with the rise of graphical user interfaces, which used objects for buttons, menus and other elements. One well-known example is Apple's Cocoa framework, used on Mac OS X and written in Objective-C. OOP toolkits also enhanced the popularity of event-driven programming.[citation needed] At ETH Zürich, Niklaus Wirth and his colleagues created new approaches to OOP. Modula-2 (1978) and Oberon (1987), included a distinctive approach to object orientation, classes, and type checking across module boundaries. Inheritance is not obvious in Wirth's design since his nomenclature looks in the opposite direction: It is called type extension and the viewpoint is from the parent down to the inheritor. Many programming languages that existed before OOP have added object-oriented features, including Ada, BASIC, Fortran, Pascal, and COBOL. This sometimes caused compatibility and maintainability issues, as these languages were not originally designed with OOP in mind. In the new millenium, new languages like Python and Ruby have emerged that combine object-oriented and procedural styles. The most commercially important "pure" object-oriented languages continue to be Java, developed by Sun Microsystems, as well as C# and Visual Basic.NET (VB.NET), both designed for Microsoft's .NET platform. These languages show the benefits of OOP by creating abstractions from implementation. The .NET platform supports cross-language inheritance, allowing programs to use objects from multiple languages together. See also: Comparison of programming languages (object-oriented programming) and List of object-oriented programming terms Object-oriented programming focuses on working with objects, but not all OOP languages have every feature linked to OOP. Below are some common features of languages that are considered strong in OOP or support it along with other programming styles. Important exceptions are also noted.[24][25][26][27] Christopher J. Date pointed out that comparing OOP with other styles, like relational programming, is difficult because there isn't a clear, agreed-upon definition of OOP.[28] Further information: Imperative programming and Structured programming Features from imperative and structured programming are present in OOP languages and are also found in non-OOP languages. Variables hold different data types like integers, strings, lists, and hash tables. Some data types are built-in while others result from combining variables using memory pointers. Procedures – also known as functions, methods, routines, or subroutines – take input, generate output, and work with data. Modern languages include structured programming constructs like loops and conditionals. Support for modular programming lets programmers organize related procedures into files and modules. This makes programs easier to manage. Each module has its own namespace, so items in one module will not conflict with items in another. Object-oriented programming (OOP) was created to make code easier to reuse and maintain.[29] However, it was not designed to clearly show the flow of a program's instructions—that was left to the compiler. As computers began using more parallel processing and multiple threads, it became more important to understand and control how instructions flow. This is difficult to do with OOP.[30][31][32][33] Main article: Object (computer science) An object is a type of data structure that has two main parts: fields and methods. Fields may also be known as members, attributes, or properties, and hold information in the form of state variables. Methods are actions, subroutines, or procedures, defining the object's behavior in code. Objects are usually stored in memory, and in many programming languages, they work like pointers that link directly to a contiguous block containing the object instances's data. Objects can contain other objects. This is called object composition. For example, an Employee object might have an Address object inside it, along with other information like "first_name" and "position". This type of structures shows "has-a" relationships, like "an employee has an address". Some believe that OOP places too much focus on using objects rather than on algorithms and data structures.[34][35] For example, programmer Rob Pike pointed out that OOP can make programmers think more about type hierarchy than composition.[36] He has called object-oriented programming "the Roman numerals of computing".[37] Rich Hickey, creator of Clojure, described OOP as overly simplistic, especially when it comes to representing real-world things that change over time.[35] Alexander Stepanov said that OOP tries to fit everything into a single type, which can be limiting. He argued that sometimes we need multisorted algebras—families of interfaces that span multiple types, such as in generic programming. Stepanov also said that calling everything an "object" doesn't add much understanding.[34] Sometimes, objects represent real-world things and processes in digital form.[38] For example, a graphics program may have objects such as "circle", "square", and "menu". An online shopping system might have objects such as "shopping cart", "customer", and "product". Niklaus Wirth said, "This paradigm [OOP] closely reflects the structure of systems in the real world and is therefore well suited to model complex systems with complex behavior".[39] However, more often, objects represent abstract entities, like an open file or a unit converter. Not everyone agrees that OOP makes it easy to copy the real world exactly or that doing so is even necessary. Bob Martin suggests that because classes are software, their relationships don't match the real-world relationships they represent.[40] Bertrand Meyer argues in Object-Oriented Software Construction, that a program is not a model of the world but a model of some part of the world; "Reality is a cousin twice removed".[41] Steve Yegge noted that natural languages lack the OOP approach of strictly prioritizing things (objects/nouns) before actions (methods/verbs), as opposed to functional programming which does the reverse.[42] This can sometimes make OOP solutions more complicated than those written in procedural programming.[43] Most OOP languages allow reusing and extending code through "inheritance". This inheritance can use either "classes" or "prototypes", which have some differences but use similar terms for ideas like "object" and "instance". In class-based programming, the most common type of OOP, every object is an instance of a specific class. The class defines the data format, like variables (e.g., name, age) and methods (actions the object can take). Every instance of the class has the same set of variables and methods. Objects are created using a special method in the class known as a constructor. Here are a few key terms in class-based OOP: Class variables – belong to the class itself, so all objects in the class share one copy. Instance variables – belong to individual objects; every object has its own version of these variables. Member variables – refers to both the class and instance variables that are defined by a particular class. Class methods – linked to the class itself and can only use class variables. Instance methods – belong to individual objects, and can use both instance and class variables Classes may inherit from other classes, creating a hierarchy of "subclasses". For example, an "Employee" class might inherit from a "Person" class. This means the Employee object will have all the variables from Person (like name variables) plus any new variables (like job position and salary). Similarly, the subclass may expand the interface with new methods. Most languages also allow the subclass to override the methods defined by superclasses. Some languages support multiple inheritance, where a class can inherit from more than one class, and other languages similarly support mixins or traits. For example, a mixin called UnicodeConversionMixin might add a method unicode_to_ascii() to both a FileReader and a WebPageScraper class. Some classes are abstract, meaning they cannot be directly instantiated into objects; they're only meant to be inherited into other classes. Other classes are utility classes which contain only class variables and methods and are not meant to be instantiated or subclassed.[44] In prototype-based programming, there aren't any classes. Instead, each object is linked to another object, called its prototype or parent. In Self, an object may have multiple or no parents,[45] but in the most popular prototype-based language, Javascript, every object has exactly one prototype link, up to the base Object type whose prototype is null. The prototype acts as a model for new objects. For example, if you have an object fruit, you can make two objects apple and orange, based on it. There is no fruit class, but they share traits from the fruit prototype. Prototype-based languages also allow objects to have their own unique properties, so the apple object might have an attribute sugar_content, while the orange or fruit objects do not. Some languages, like Go, don't use inheritance at all.[46] Instead, they encourage "composition over inheritance", where objects are built using smaller parts instead of parent-child relationships. For example, instead of inheriting from class Person, the Employee class could simply contain a Person object. This lets the Employee class control how much of Person it exposes to other parts of the program. Delegation is another language feature that can be used as an alternative to inheritance. Programmers have different opinions on inheritance. Bjarne Stroustrup, author of C++, has stated that it is possible to do OOP without inheritance.[47] Rob Pike has criticized inheritance for creating complicated hierarchies instead of simpler solutions.[48] See also: Object-oriented design People often think that if one class inherits from another, it means the subclass "is a" more specific version of the original class. This presumes the program semantics are that objects from the subclass can always replace objects from the original class without problems. This concept is known as behavioral subtyping, more specifically the Liskov substitution principle. However, this is often not true, especially in programming languages that allow mutable objects, objects that change after they are created. In fact, subtype polymorphism as enforced by the type checker in OOP languages cannot guarantee behavioral subtyping in most if not all contexts. For example, the circle-ellipse problem is notoriously difficult to handle using OOP's concept of inheritance. Behavioral subtyping is undecidable in general, so it cannot be easily implemented by a compiler. Because of this, programmers must carefully design class hierarchies to avoid mistakes that the programming language itself cannot catch. When a method is called on an object, the object itself—not outside code—decides which specific code to run. This process, called dynamic dispatch, usually happens at run time by checking a table linked to the object to find the correct method. In this context, a method call is also known as message passing, meaning the method name and its inputs are like a message sent to the object for it to act on. If the method choice depends on more than one type of object (such as other objects passed as parameters), it's called multiple dispatch. Dynamic dispatch works together with inheritance: if an object doesn't have the requested method, it looks up to its parent class (delegation), and continues up the chain until it finds the method or reaches the top. Data abstraction is a way of organizing code so that only certain parts of the data are visible to related functions (data hiding). This helps prevent mistakes and makes the program easier to manage. Because data abstraction works well, many programming styles, like object-oriented programming and functional programming, use it as a key principle. Encapsulation is another important idea in programming. It means keeping the internal details of an object hidden from the outside code. This makes it easier to change how an object works on the inside without affecting other parts of the program, such as in code refactoring. Encapsulation also helps keep related code together (decoupling), making it easier for programmers to understand. In object-oriented programming, objects act as a barrier between their internal workings and external code. Outside code can only interact with an object by calling specific public methods or variables. If a class only allows access to its data through methods and not directly, this is called information hiding. When designing a program, it's often recommended to keep data as hidden as possible. This means using local variables inside functions when possible, then private variables (which only the object can use), and finally public variables (which can be accessed by any part of the program) if necessary. Keeping data hidden helps prevent problems when changing the code later.[49] Some programming languages, like Java, control information hiding by marking variables as private (hidden) or public (accessible).[50] Other languages, like Python, rely on naming conventions, such as starting a private method's name with an underscore. Intermediate levels of access also exist, such as Java's protected keyword, (which allows access from the same class and its subclasses, but not objects of a different class), and the internal keyword in C#, Swift, and Kotlin, which restricts access to files within the same module.[51] Abstraction and information hiding are important concepts in programming, especially in object-oriented languages.[52] Programs often create many copies of objects, and each one works independently. Supporters of this approach say it makes code easier to reuse and intuitively represents real-world situations.[53] However, others argue that object-oriented programming does not enhance readability or modularity.[54][55] Eric S. Raymond has written that object-oriented programming languages tend to encourage thickly layered programs that destroy transparency.[56] Raymond compares this unfavourably to the approach taken with Unix and the C programming language.[56] One programming principle, called the "open/closed principle", says that classes and functions should be "open for extension, but closed for modification". Luca Cardelli has stated that OOP languages have "extremely poor modularity properties with respect to class extension and modification", and tend to be extremely complex.[54] The latter point is reiterated by Joe Armstrong, the principal inventor of Erlang, who is quoted as saying:[55] The problem with object-oriented languages is they've got all this implicit environment that they carry around with them. You wanted a banana but what you got was a gorilla holding the banana and the entire jungle. Leo Brodie says that information hiding can lead to copying the same code in multiple places (duplicating code),[57] which goes against the don't repeat yourself rule of software development.[58] Polymorphism is the use of one symbol to represent multiple different types.[59] In object-oriented programming, polymorphism more specifically refers to subtyping or subtype polymorphism, where a function can work with a specific interface and thus manipulate entities of different classes in a uniform manner.[60] For example, imagine a program has two shapes: a circle and a square. Both come from a common class called "Shape." Each shape has its own way of drawing itself. With subtype polymorphism, the program doesn't need to know the type of each shape, and can simply call the "Draw" method for each shape. The programming language runtime will ensure the correct version of the "Draw" method runs for each shape. Because the details of each shape are handled inside their own classes, this makes the code simpler and more organized, enabling strong separation of concerns. In object-oriented programming, objects have methods that can change or use the object's data. Many programming languages use a special word, like this or self, to refer to the current object. In languages that support open recursion, a method in an object can call other methods in the same object, including itself, using this special word. This allows a method in one class to call another method defined later in a subclass, a feature known as late binding. This section does not cite any sources. Please help improve this section by adding citations to reliable sources. Unsourced material may be challenged and removed. (August 2009) (Learn how and when to remove this message) See also: List of object-oriented programming languages OOP languages can be grouped into different types based on how they support and use objects: Pure OOP languages: In these languages, everything is treated as an object, even basic things like numbers and characters. They are designed to fully support and enforce OOP. Examples: Ruby, Scala, Smalltalk, Eiffel, Emerald,[61] JADE, Self, Raku. Mostly OOP languages: These languages focus on OOP but also include some procedural programming features. Examples: Java, Python, C++, C#, Delphi/Object Pascal, VB.NET. Retrofitted OOP languages: These were originally designed for other types of programming but later added some OOP features. Examples: PHP, JavaScript, Perl, Visual Basic (derived from BASIC), MATLAB, COBOL 2002, Fortran 2003, ABAP, Ada 95, Pascal. Unique OOP languages: These languages have OOP features like classes and inheritance but use them in their own way. Examples: Oberon, BETA. Object-based languages: These support some OOP ideas but avoid traditional class-based inheritance in favor of direct manipulation of objects. Examples: JavaScript, Lua, Modula-2, CLU, Go. Multi-paradigm languages: These support both OOP and other programming styles, but OOP is not the predominant style in the language. Examples include Tcl, where TclOO allows both prototype-based and class-based OOP, and Common Lisp, with its Common Lisp Object System. The TIOBE programming language popularity index graph from 2002 to 2023. In the 2000s the object-oriented Java (orange) and the procedural C (dark blue) competed for the top position. Many popular programming languages, like C++, Java, and Python, use object-oriented programming. In the past, OOP was widely accepted,[62] but recently, some programmers have criticized it and prefer functional programming instead.[63] A study by Potok et al. found no major difference in productivity between OOP and other methods.[64] Paul Graham, a well-known computer scientist, believes big companies like OOP because it helps manage large teams of average programmers. He argues that OOP adds structure, making it harder for one person to make serious mistakes, but at the same time restrains smart programmers.[65] Eric S. Raymond, a Unix programmer and open-source software advocate, argues that OOP is not the best way to write programs.[56] Richard Feldman says that, while OOP features helped some languages stay organized, their popularity comes from other reasons.[66] Lawrence Krubner argues that OOP doesn't offer special advantages compared to other styles, like functional programming, and can make coding more complicated.[67] Luca Cardelli says that OOP is slower and takes longer to compile than procedural programming.[54] In recent years, object-oriented programming (OOP) has become very popular in dynamic programming languages. Some languages, like Python, PowerShell, Ruby and Groovy, were designed with OOP in mind. Others, like Perl, PHP, and ColdFusion, started as non-OOP languages but added OOP features later (starting with Perl 5, PHP 4, and ColdFusion version 6). On the web, HTML, XHTML, and XML documents use the Document Object Model (DOM), which works with the JavaScript language. JavaScript is a well-known example of a prototype-based language. Instead of using classes like other OOP languages, JavaScript creates new objects by copying (or "cloning") existing ones. Another language that uses this method is Lua. When computers communicate in a client-server system, they send messages to request services. For example, a simple message might include a length field (showing how big the message is), a code that identifies the type of message, and a data value. These messages can be designed as structured objects that both the client and server understand, so that each type of message corresponds to a class of objects in the client and server code. More complex messages might include structured objects as additional details. The client and server need to know how to serialize and deserialize these messages so they can be transmitted over the network, and map them to the appropriate object types. Both clients and servers can be thought of as complex object-oriented systems. The Distributed Data Management Architecture (DDM) uses this idea by organizing objects into four levels: Basic message details - Information like message length, type, and data. Objects and collections - Similar to how objects work in Smalltalk, storing messages and their details. Managers - Like file directories, these organize and store data, as well as provide memory and processing power. They are similar to IBM i Objects. Clients and servers - These are full systems that include managers and handle security, directory services, and multitasking. The first version of DDM defined distributed file services. Later, it was expanded to support databases through the Distributed Relational Database Architecture (DRDA). Design patterns are common solutions to problems in software design. Some design patterns are especially useful for object-oriented programming, and design patterns are typically introduced in an OOP context. The following are notable software design patterns for OOP objects.[68] Function object: Class with one main method that acts like an anonymous function (in C++, the function operator, operator()) Immutable object: does not change state after creation First-class object: can be used without restriction Container object: contains other objects Factory object: creates other objects Metaobject: Used to create other objects (similar to a class, but an object) Prototype object: a specialized metaobject that creates new objects by copying itself Singleton object: only instance of its class for the lifetime of the program Filter object: receives a stream of data as its input and transforms it into the object's output A common anti-pattern is the God object, an object that knows or does too much. Main article: Design pattern (computer science) Design Patterns: Elements of Reusable Object-Oriented Software is a famous book published in 1994 by four authors: Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. People often call them the "Gang of Four". The book talks about the strengths and weaknesses of object-oriented programming and explains 23 common ways to solve programming problems. These solutions, called "design patterns," are grouped into three types: Creational patterns (5): Factory method pattern, Abstract factory pattern, Singleton pattern, Builder pattern, Prototype pattern Structural patterns (7): Adapter pattern, Bridge pattern, Composite pattern, Decorator pattern, Facade pattern, Flyweight pattern, Proxy pattern Behavioral patterns (11): Chain-of-responsibility pattern, Command pattern, Interpreter pattern, Iterator pattern, Mediator pattern, Memento pattern, Observer pattern, State pattern, Strategy pattern, Template method pattern, Visitor pattern Main articles: Object-relational impedance mismatch, Object-relational mapping, and Object database Both object-oriented programming and relational database management systems (RDBMSs) are widely used in software today. However, relational databases don't store objects directly, which creates a challenge when using them together. This issue is called object-relational impedance mismatch. To solve this problem, developers use different methods, but none of them are perfect.[69] One of the most common solutions is object-relational mapping (ORM), which helps connect object-oriented programs to relational databases. Examples of ORM tools include Visual FoxPro, Java Data Objects, and Ruby on Rails ActiveRecord. Some databases, called object databases, are designed to work with object-oriented programming. However, they have not been as popular or successful as relational databases. Date and Darwen have proposed a theoretical foundation that uses OOP as a kind of customizable type system to support RDBMSs, but it forbids objects containing pointers to other objects.[70] In responsibility-driven design, classes are built around what they need to do and the information they share, in the form of a contract. This is different from data-driven design, where classes are built based on the data they need to store. According to Wirfs-Brock and Wilkerson, the originators of responsibility-driven design, responsibility-driven design is the better approach.[71] SOLID is a set of five rules for designing good software, created by Michael Feathers: Single responsibility principle: A class should have only one reason to change. Open/closed principle: Software entities should be open for extension, but closed for modification. Liskov substitution principle: Functions that use pointers or references to base classes must be able to use objects of derived classes without knowing it. Interface segregation principle: Clients should not be forced to depend upon interfaces that they do not use. Dependency inversion principle: Depend upon abstractions, not concretes. GRASP (General Responsibility Assignment Software Patterns) is another set of software design rules, created by Craig Larman, that helps developers assign responsibilities to different parts of a program:[72] Creator Principle: allows classes create objects they closely use. Information Expert Principle: assigns tasks to classes with the needed information. Low Coupling Principle: reduces class dependencies to improve flexibility and maintainability. High Cohesion Principle: designing classes with a single, focused responsibility. Controller Principle: assigns system operations to separate classes that manage flow and interactions. Polymorphism: allows different classes to be used through a common interface, promoting flexibility and reuse. Pure Fabrication Principle: create helper classes to improve design, boost cohesion, and reduce coupling. See also: Formal semantics of programming languages In object-oriented programming, objects are things that exist while a program is running. An object can represent anything, like a person, a place, a bank account, or a table of data. Many researchers have tried to formally define how OOP works. Records are the basis for understanding objects. They can represent fields, and also methods, if function literals can be stored. However, inheritance presents difficulties, particularly with the interactions between open recursion and encapsulated state. Researchers have used recursive types and co-algebraic data types to incorporate essential features of OOP.[73] Abadi and Cardelli defined several extensions of System F