Programming Prerequisites Cheatsheet 🚀

From Complete Newbie to Advance Developer

Level 1: Absolute Beginner Prerequisites 🌱

1. **Basic Computer Literacy** 💻

- **Know how to:**
  - Use a keyboard and mouse efficiently
  - Install and uninstall applications
  - Browse files and folders (understand file paths like `C:\Users\YourName\Documents`)
  - Use a browser and search for help
  - Take screenshots and copy/paste text
  - Understand file extensions (.txt, .jpg, .exe, .py, .js)

2. **Typing Skills** ⌨️

- **Practice using:**
  - keybr.com or typingclub.com
  - Learn to type with both hands (at least 30 WPM)
  - **Important symbols:** `{} [] () ; : " ' < > / \ | & * + - = _`
  - **Real-life example:** In Python, you'll type `print("Hello World")` - notice the parentheses, quotes, and exact spacing!

3. **Understanding How Computers Work** ⚙️

- **Hardware Deep Dive:**
  - **CPU (Central Processing Unit):** The "brain" that executes instructions one by one. Modern CPUs can execute billions of instructions per second. Think of it as a super-fast calculator that can perform basic operations like addition, comparison, and moving data around.
  - **RAM (Random Access Memory):** Temporary storage that holds data while programs are running. When you open a program, it gets loaded from storage into RAM because RAM is much faster to access. It's volatile - everything disappears when power is lost.
  - **Storage (Hard Drive/SSD):** Permanent storage that keeps data even when computer is off. HDDs use spinning disks (slower, cheaper), SSDs use electronic chips (faster, more expensive).
  - **Input/Output devices:** How humans interact with computers - keyboard, mouse, touchscreen (input) and monitor, speakers, printer (output).
- **Software Architecture:**
  - **Operating System (OS):** The master program that manages all other programs and hardware. Windows, macOS, Linux are different ways of organizing and controlling your computer.
  - **Applications vs System Software:** Apps are programs you use (browser, games, text editors), system software manages the computer itself (drivers, utilities).
  - **Files, Folders, and Programs:** Files are individual pieces of data, folders organize files, programs are special files that contain instructions the CPU can execute.

- **How It All Works Together:** When you click a program icon, the OS finds the program file on storage, loads it into RAM, and tells the CPU to start executing its instructions. The program can then read input from keyboard/mouse and display output on screen.
- **Real-life analogy:** Your computer is like a kitchen - CPU is the chef who follows recipes (programs), RAM is the counter space for current ingredients (active data), storage is the pantry for long-term ingredients (files), and the OS is like the kitchen manager coordinating everything.

4. **Basic English Reading Skills** 📖

- Most programming languages use English keywords (`if`, `while`, `function`, `class`)
- Documentation, error messages, and tutorials are primarily in English
- **Example:** `if (age >= 18) { console.log("You can vote!"); }` - notice English words

5. **Math Basics** ➕➖✖️➗

- **Essential math (no advanced calculus needed!):**
  - Basic arithmetic: +, -, ×, ÷, % (modulo)
  - Logical thinking: if-then relationships
  - Pattern recognition
  - **Optional but helpful:** algebra variables, basic geometry
- **Real-life example:** `total = price * quantity + tax` - this is just basic math!

Level 2: Foundation Skills 🏗️

6. **Google Search & Research Skills** 🔍

- **Learn to search for:**
  - Error messages: `"TypeError: Cannot read property" JavaScript`
  - Tutorials: `"how to create a loop in Python for beginners"`
  - Documentation: `"Python string methods official docs"`
- **Essential websites:**
  - Stack Overflow (Q&A for programmers)
  - MDN Web Docs (web development)
  - W3Schools (web tutorials)
  - Official language documentation

7. **Problem-Solving & Logic** 🧠

- **Practice logical thinking with:**
  - Puzzles: Sudoku, crosswords, logic grid puzzles
  - Games: Chess, programming games like Human Resource Machine
  - **Websites:** brilliant.org, codecademy.com logic courses
- **Real-life example:** "If it's raining AND I don't have an umbrella, THEN I'll get wet" = programming logic!

8. **Code Editor & Development Environment** 👨‍💻

- **Start with beginner-friendly editors:**

  - VS Code (most popular, free)
  - Sublime Text
  - Atom (discontinued but still usable)
- **Learn basic editor skills:**
  - Create and save files
  - Use syntax highlighting
  - Install extensions/plugins
  - Use find and replace
- **Real-life example:** Writing code is like writing a document - you need a good word processor (code editor)

9. **Understanding What Programming Actually Is** 💡

- **Programming is NOT:**
  - Memorizing syntax
  - Being a "computer genius"
  - Only for math wizards
- **Programming IS:**
  - Giving step-by-step instructions to solve problems
  - Breaking big problems into smaller pieces
  - Automating repetitive tasks
  - Creating tools that help people
- **Real-life analogy:** It's like writing a recipe - you give precise instructions so anyone (or any computer) can follow them

---

Level 3: Programming Fundamentals 🎯

10. **Choose Your First Programming Language** 🗣️

- **For absolute beginners:**
  - **Python** - easiest to read, great for beginners
  - **JavaScript** - runs in web browsers, instant results
  - **Scratch** - visual programming, no typing required
- **Real-life examples:**

```python
# Python - looks almost like English
if temperature > 30:
    print("It's hot outside!")
```

```javascript
// JavaScript - powers websites
if (age >= 18) {
    alert("You can vote!");
}
```

11. **Basic Programming Concepts** 💻

- **Variables (storing information):**
  - `name = "John"` (like putting a label on a box)
  - `age = 25`
  - `is_student = True`
- **Data types:**
  - Text (strings): `"Hello"`
  - Numbers: `42`, `3.14`
  - True/False (booleans): `True`, `False`
- **Real-life example:** Variables are like labeled containers - you store milk in a container labeled "milk"

12. **Basic Operations & Logic** 🔄

- **Arithmetic:** `+`, `-`, `*`, `/`, `%`
- **Comparison:** `>`, `<`, `==`, `!=`
- **Logic:** `and`, `or`, `not`
- **Real-life example:**

```
price = 100
discount = 20
final_price = price - discount
if final_price < 50:
    print("Great deal!")
```

Level 4: Intermediate Prerequisites 📈

13. **Control Structures (Making Decisions)** 🚦

- **If statements:** Making choices

```
if weather == "sunny":
    print("Go to the beach!")
elif weather == "rainy":
    print("Stay inside")
else:
    print("Check weather app")
```

- **Loops:** Repeating actions

```
# Print numbers 1 to 5
for number in range(1, 6):
    print(number)
```

14. **Functions (Reusable Code Blocks)** 🔧

- **What are functions:** Like recipes you can use over and over

```
def greet_person(name):
    return f"Hello, {name}!"

# Use the function
message = greet_person("Alice")
print(message)  # Output: Hello, Alice!
```

- **Real-life analogy:** Functions are like appliances - a toaster always makes toast, no matter what bread you put in

15. **Data Structures (Organizing Information)** 📊

- **Lists/Arrays:** Like a shopping list

```
fruits = ["apple", "banana", "orange"]
print(fruits[0])  # Output: apple
```

- **Dictionaries/Objects:** Like a phone book

```
person = {
    "name": "John",
    "age": 30,
    "city": "New York"
}
print(person["name"])  # Output: John
```

16. **File Handling** 🗀

- **Reading files:** Getting information from documents
- **Writing files:** Saving information to documents
- **Real-life example:**

```
# Read a grocery list from a file
with open("groceries.txt", "r") as file:
    items = file.read()
    print(items)
```

Level 5: Advanced Prerequisites 🎓

17. **Object-Oriented Programming (OOP)** 🏗

- **Classes and Objects:** Like blueprints and houses

```python
class Car:
    def __init__(self, brand, color):
        self.brand = brand
        self.color = color

    def start_engine(self):
        print(f"The {self.color} {self.brand} is starting!")

my_car = Car("Toyota", "red")
my_car.start_engine()  # Output: The red Toyota is starting!
```

- **Real-life analogy:** A class is like a cookie cutter, objects are the actual cookies

18. **Error Handling** 🚨

- **Try-catch blocks:** Handling when things go wrong

```python
try:
    age = int(input("Enter your age: "))
    print(f"You are {age} years old")
except ValueError:
    print("Please enter a valid number")
```

- **Real-life example:** Like having a backup plan when your original plan fails

19. **APIs and Libraries** 📚

- **Libraries:** Pre-written code you can use

```python
import requests  # Library for web requests

response = requests.get("https://api.weather.com/current")
weather_data = response.json()
```

- **APIs:** Ways for programs to talk to each other
- **Real-life analogy:** Libraries are like using a calculator instead of doing math by hand

20. **Version Control (Git)** 🗃

- **What is Git:** Keeping track of changes in your code
- **Basic commands:**
  - `git add .` - Stage changes

- `git commit -m "message"` - Save changes
- `git push` - Upload to internet

- **Real-life analogy:** Like "Save As" but for programmers, with a complete history

---

Level 6: System Design & Computer Science 🔬

21. **Algorithms & Data Structures (DSA)** 🧮

- **What are Algorithms:** Step-by-step procedures for solving problems efficiently. Every task you do has an algorithm - like following a recipe to cook, or the steps you take to find a book in a library. In programming, algorithms help us solve problems like sorting data, finding shortest paths, or searching through millions of records quickly.
- **Why DSA Matters:** Different algorithms can solve the same problem with vastly different speeds. Imagine sorting 1 million numbers - a bad algorithm might take hours, while a good one takes seconds. Companies like Google, Facebook, and Amazon rely on efficient algorithms to handle billions of users. This is why technical interviews focus heavily on DSA.
- **Essential Data Structures:**
  - **Arrays:** Fixed-size lists where elements are stored in consecutive memory locations. Great for random access (like jumping to page 50 in a book) but bad for insertions in the middle.
  - **Linked Lists:** Elements connected by pointers, like a treasure hunt where each clue leads to the next. Good for insertions but slow for finding specific elements.
  - **Trees:** Hierarchical structures like family trees or company org charts. Binary search trees allow super-fast searching, sorting, and insertion.
  - **Graphs:** Networks of connected nodes, like social media connections or city road maps. Used in GPS navigation, social networks, and recommendation systems.
  - **Hash Tables:** Like a super-efficient filing system where you can instantly find any document by its name. Used in dictionaries, database indexing, and caching.
- **Common Algorithm Types:**
  - **Sorting:** Organizing data (bubble sort, merge sort, quicksort). Like organizing your music library alphabetically.
  - **Searching:** Finding specific items (binary search, depth-first search). Like finding a word in a dictionary.
  - **Dynamic Programming:** Breaking complex problems into smaller subproblems and remembering solutions. Like solving a jigsaw puzzle by working on small sections.
  - **Recursion:** Functions that call themselves to solve smaller versions of the same problem. Like Russian nesting dolls - each doll contains a smaller version of itself.
- **Real-world Applications:** Netflix uses recommendation algorithms to suggest movies, GPS uses shortest-path algorithms for directions, Google uses PageRank algorithm for search results, databases use sorting algorithms for query optimization.
- **Time Complexity (Big O):** How algorithm performance changes as input size grows. O(1) = constant time (like looking up a phone number), O(log n) = logarithmic (like binary search in dictionary), O(n) = linear (like reading every page of a book), O(n^2) = quadratic (like comparing every person with every other person in a room).

22. **Database Management Systems (DBMS)** 🗄️

- **What are Databases:** Organized collections of structured data that can be easily accessed, managed, and updated. Unlike simple files, databases provide powerful ways to store millions of records, ensure data integrity, handle multiple users simultaneously, and perform complex queries across related data. Think of it as the difference between keeping customer information in a notebook versus a sophisticated filing system with cross-references and instant search capabilities.
- **Why Databases Matter:** Every app you use (Instagram, banking apps, online shopping) relies on databases to store user accounts, posts, transactions, and product catalogs. Without databases, every time you opened an app, you'd have to re-enter all your information. Databases provide persistence, reliability, and scalability that simple file storage cannot match.
- **Types of Databases:**
  - **Relational Databases (SQL):** Data stored in tables with rows and columns, like Excel spreadsheets that can be connected. Examples: MySQL, PostgreSQL, Oracle. Great for structured data with clear relationships (customer orders, employee records).
  - **NoSQL Databases:** Flexible storage for unstructured data like social media posts, images, or sensor data. Examples: MongoDB, Firebase, Redis. Great for rapid development and handling varied data types.
  - **Graph Databases:** Specialized for data with complex relationships, like social networks or recommendation engines. Examples: Neo4j, Amazon Neptune.
- **Database Operations (CRUD):**
  - **Create:** Adding new data (`INSERT INTO users VALUES ('John', 25)`)
  - **Read:** Retrieving data (`SELECT name FROM users WHERE age > 18`)
  - **Update:** Modifying existing data (`UPDATE users SET age = 26 WHERE name = 'John'`)
  - **Delete:** Removing data (`DELETE FROM users WHERE name = 'John'`)
- **Key Concepts:**
  - **Tables/Collections:** Organized containers for similar data (all user accounts in one table)
  - **Primary Keys:** Unique identifiers for each record (like social security numbers)
  - **Foreign Keys:** Links between related tables (connecting user accounts to their orders)
  - **Indexes:** Speed up searches by creating shortcuts (like index in a textbook)
  - **Transactions:** Ensuring multiple operations succeed or fail together (transferring money requires both debit and credit to complete)
- **Real-world Applications:** E-commerce sites use databases to track inventory, process orders, and manage customer accounts. Banks use databases for account balances and transaction history with strict security and backup requirements. Social media platforms use databases to store posts, relationships, and user preferences at massive scale.
- **Database Design Principles:** Normalization (avoiding duplicate data), ACID properties (Atomicity, Consistency, Isolation, Durability) for reliable transactions, and proper indexing for performance optimization.

23. Networking & Web Development Fundamentals 🌐

- **How the Internet Works:** The internet is a global network of interconnected computers that communicate using standardized protocols. When you visit a website, your computer (client) sends a request through your Internet Service Provider (ISP) to servers hosting that website. The request travels through multiple routers and networks before reaching the destination server, which then sends back the website data.
- **Core Networking Concepts:**

- **HTTP/HTTPS:** HyperText Transfer Protocol - the language browsers use to request web pages from servers. HTTPS adds encryption for security. Like the difference between sending a postcard (HTTP) vs a sealed letter (HTTPS).
- **DNS (Domain Name System):** Translates human-readable website names (google.com) into IP addresses (172.217.164.110) that computers use to locate servers. Like a phone book for the internet.
- **IP Addresses & Ports:** Every device on the internet has a unique IP address (like a postal address). Ports are like apartment numbers - they specify which service on a server to connect to (port 80 for web, port 25 for email).
- **Client-Server Model:** Your device (client) requests services from remote computers (servers). Servers are powerful computers designed to handle multiple simultaneous requests from many clients.

- **Web Development Architecture:**
  - **Frontend (Client-Side):** Code that runs in the user's browser - HTML for structure, CSS for styling, JavaScript for interactivity. Users directly interact with frontend code.
  - **Backend (Server-Side):** Code that runs on servers - handles business logic, database operations, user authentication, and sends data to frontend. Users never see backend code directly.
  - **APIs (Application Programming Interfaces):** Standardized ways for different software systems to communicate. REST APIs use HTTP requests to exchange data, like standardized forms for requesting information.
- **Web Request Lifecycle:**
  1. User types URL or clicks link
  2. Browser looks up server IP address via DNS
  3. Browser sends HTTP request to server
  4. Server processes request (may query database, run business logic)
  5. Server sends back HTTP response with data
  6. Browser receives data and renders webpage
  7. JavaScript may make additional API calls for dynamic content
- **Modern Web Technologies:**
  - **Single Page Applications (SPAs):** Web apps that load once and update content dynamically without full page reloads. Like mobile apps but running in browsers.
  - **Progressive Web Apps (PWAs):** Web applications that work offline and can be installed like native mobile apps.
  - **WebSockets:** Real-time, two-way communication between client and server. Used for chat applications, live updates, online gaming.
- **Real-world Applications:** When you use Gmail, the frontend (what you see) runs in your browser, but the backend (email storage, spam filtering, search) runs on Google's servers. When you send an email, your browser makes an API call to Google's servers, which process the request and store/forward your message.

24. **System Architecture & Design** 🏢

- **What is System Design:** The process of defining system components, modules, architecture, interfaces, and data flow to satisfy specified requirements. It's about creating blueprints for large-scale applications that can handle millions of users, process massive amounts of data, and remain reliable under heavy load. System design bridges the gap between theoretical programming knowledge and real-world scalable applications.

- **Why System Design Matters:** When Netflix serves 200+ million users simultaneously, or when WhatsApp handles billions of messages daily, it's not just about writing good code - it's about designing systems that can scale horizontally (adding more servers), handle failures gracefully, and maintain performance under extreme load. Poor system design leads to applications that crash during high traffic, lose user data, or become too expensive to maintain.
- **Core System Design Concepts:**
  - **Scalability:** Horizontal (adding more servers) vs Vertical (upgrading existing servers). Like hiring more cashiers vs training one cashier to work faster.
  - **Load Balancing:** Distributing incoming requests across multiple servers to prevent any single server from being overwhelmed. Like having multiple checkout lanes at a grocery store.
  - **Caching:** Storing frequently accessed data in fast storage to reduce database load. Like keeping bestselling books at the front of a bookstore.
  - **Database Sharding:** Splitting large databases across multiple servers. Like having separate filing cabinets for customers A-M and N-Z.
  - **Microservices vs Monoliths:** Breaking large applications into smaller, independent services vs keeping everything in one large application. Like having specialized restaurants (pizza, burgers, sushi) vs one restaurant serving everything.
- **System Design Patterns:**
  - **Client-Server Architecture:** Separation between user interface (client) and data processing (server). Your phone app talks to company servers.
  - **Publisher-Subscriber:** Components communicate through message queues without direct connections. Like newspaper subscription - publishers don't need to know individual subscribers.
  - **Master-Slave Replication:** One primary database handles writes, multiple replicas handle reads. Ensures data availability and improves read performance.
  - **Content Delivery Networks (CDNs):** Storing copies of static content (images, videos) closer to users geographically. Like having local warehouses instead of shipping everything from one central location.
- **Real-world System Design Examples:**
  - **Social Media Platform:** User service (profiles), post service (content), notification service (alerts), media service (photos/videos), recommendation service (feeds) - all working together but independently scalable.
  - **E-commerce Platform:** Product catalog service, inventory management, order processing, payment gateway, shipping service, recommendation engine - each can be scaled based on demand.
  - **Video Streaming Service:** Content delivery network for videos, user authentication, recommendation algorithms, encoding services for different video qualities, analytics for viewing patterns.
- **System Design Interview Topics:** Designing systems like Twitter, Uber, YouTube, or WhatsApp - focusing on scalability, reliability, consistency, and performance trade-offs rather than specific code implementation.

Level 7: Professional Development 💼

25. **Testing & Debugging** 🔍

- **Unit testing:** Making sure individual parts work

- **Debugging:** Finding and fixing problems
- **Real-life example:** Testing each ingredient before cooking a meal

26. **Security Basics** 🔒

- **Common vulnerabilities:** SQL injection, XSS attacks
- **Best practices:** Never trust user input, encrypt sensitive data
- **Real-life analogy:** Like locking your house and not giving keys to strangers

27. **Performance & Optimization** ⚡

- **Big O notation:** Understanding how fast your code runs
- **Caching:** Storing frequently used data for quick access
- **Real-life example:** Memorizing your friend's phone number instead of looking it up each time

28. **Development Methodologies** 📋

- **Agile/Scrum:** Working in short cycles with regular check-ins
- **DevOps:** Combining development and operations
- **Code reviews:** Having other programmers check your work

Practical Next Steps 🎯

Month 1-2: Start Here

1. Learn basic computer skills and typing
2. Choose Python or JavaScript as first language
3. Complete online tutorials (Codecademy, freeCodeCamp)
4. Build simple projects (calculator, to-do list)

Month 3-6: Build Foundation

1. Learn Git and GitHub
2. Build 5-10 small projects
3. Join programming communities (Reddit r/learnprogramming)
4. Start reading other people's code

Month 6-12: Intermediate Skills

1. Learn a web framework (React, Django, Express)
2. Build a complete web application
3. Learn database basics
4. Contribute to open-source projects

Year 2+: Advanced Topics

1. System design principles
2. Advanced algorithms and data structures
3. Specialization (web dev, mobile, AI, etc.)

4. Build portfolio and apply for jobs

Essential Resources 📚

Free Learning Platforms:

- **Codecademy** - Interactive coding lessons
- **freeCodeCamp** - Comprehensive web development curriculum
- **Khan Academy** - Computer programming basics
- **CS50** - Harvard's intro to computer science

Practice Platforms:

- **LeetCode** - Coding challenges
- **HackerRank** - Programming competitions
- **Codepen** - Frontend web development playground
- **Repl.it** - Online coding environment

Communities:

- **Reddit:** r/learnprogramming, r/webdev, r/Python
- **Discord:** Various programming servers
- **Stack Overflow** - Q&A for specific problems
- **GitHub** - Code repositories and collaboration

Remember: Programming is a Journey, Not a Destination! 🌟

- **Start small:** Even senior developers started with "Hello World"
- **Practice consistently:** 30 minutes daily is better than 5 hours once a week
- **Don't be afraid to make mistakes:** Every bug is a learning opportunity
- **Build projects:** Apply what you learn to real problems
- **Join communities:** Programming is collaborative, not solitary
- **Stay curious:** Technology changes fast, but fundamentals remain the same

Happy coding! 🚀👩‍💻👨‍💻