# 100-Days of code Syllabus: Day-by-Day Topics

## Part 1: Python Basics & Core Concepts (Days 1-24)

**Day 1: Medical History Snippet Generator**

- The `print()` function
- String Manipulation & Concatenation
- The `input()` function
- Variables
- Debugging: Identifying `SyntaxError`
- PEP 8 & Semantic Naming Conventions
- String Formatting (f-strings)

**Day 2: Simple BMI Calculator**

- Data Types: Strings, Integers, Floats
- Type Checking ( `type()` function)
- Type Conversion ( `int()` , `float()` , `str()` )
- Mathematical Operators ( `+` , `-` , `*` , `/` , `**` )
- f-Strings & String Formatting
- Type Hinting (e.g., `weight: float` )

**Day 3: Simple Symptom Triage Bot**

- Conditional Statements: `if` , `elif` , `else`
- Comparison Operators ( `==` , `!=` , `>` , `<` , `>=` , `<=` )
- Logical Operators: `and` , `or` , `not`
- Nested `if` statements
- String Methods: `.lower()` and `.upper()`
- Guard Clauses (as an alternative to deep nesting)

**Day 4: NEET Question Shuffler**

- `random` Module: `random.randint()` , `random.choice()` , `random.shuffle()`
- Python Lists: Creating, appending, and extending
- List Indexing: 0-based and negative indexing

- Data Structures: Lists vs. Tuples

**Day 5: Average Student Score Calculator**

- `for` Loops
- Looping through Lists
- The `range()` function
- `len()` function
- `sum()` function
- String Methods: `.split()`
- Type Conversion within loops
- List Comprehensions (as an advanced alternative)

**Day 6: A Simple "Hydration Reminder"**

- Functions: Defining ( `def` ) and Calling
- `while` Loops
- Creating Exit Conditions & Avoiding Infinite Loops
- The DRY (Don't Repeat Yourself) Principle
- Variable Incrementors ( `+=` )

**Day 7: Simple Word Guessing Game (Code Breaker)**

- Project Synthesis: Combining Loops, Conditionals, and Lists
- State Management: Using variables to track game state (e.g., `display` , `game_over` )
- `for` loops with `range(len(list))` for index-based modification
- String/List Comparison
- The `in` keyword

**Day 8: Prescription Dosage Calculator**

- Functions with Inputs: Parameters and Arguments
- Positional vs. Keyword Arguments
- Functions with Default Values
- Local Scope (variables inside functions)

**Day 9: Patient Record Keeper**

- Dictionaries: Key-Value Pairs
- Accessing and Modifying Dictionary items
- Looping through Dictionaries
- Nesting: Dictionaries in Dictionaries, Lists in Dictionaries

- The `.get()` method for safe dictionary access

**Day 10: Lab Result Formatter**

- Functions with Outputs: The `return` keyword
- Functions with multiple `return` statements
- Python Tuples (for `normal_range`)
- Writing Docstrings (documenting functions)
- Command-Query Separation principle

**Day 11: The Blackjack Capstone Project**

- Problem Decomposition: Breaking a large project into small functions
- Global vs. Local Scope
- Global Constants
- Using `return` values as inputs for other functions
- Handling complex logic with nested conditionals

**Day 12: Health Score Calculator with Scope**

- Local Scope (inside functions) vs. Global Scope (outside functions)
- Namespaces
- Why modifying global variables is bad practice
- The "Pass-in and Return" pattern for state modification
- Block Scope (and Python's lack of it)

**Day 13: Debug the BMI Formula**

- The Debugging Mindset: Code does what you *tell* it, not what you *want* it to do
- Reading Tracebacks (Error Messages)
- Common Error Types: `TypeError`, `NameError`
- Using `print()` statements to trace variable values
- Fixing Logical Errors (vs. Syntax Errors)
- Using an IDE's visual debugger (Breakpoints, Step Over, Step Into)

**Day 14: Disease Prevalence Game**

- Managing and updating game state in a loop
- Accessing data from a list of dictionaries
- `random.choice()`
- Function decomposition
- Clear User Feedback (f-strings)

## Day 15: Pharmacy Inventory Manager

- Resource Management Simulation
- Using dictionaries as a data store (for inventory, prices)
- Decomposing logic into functions (e.g., `check_resources` , `process_payment` )
- Looping with `while True:` and `break`
- Handling user input within a loop

## Day 16: Define a MedicalTest Class

- Object-Oriented Programming (OOP) Concepts
- Creating a `class`
- The `__init__()` Constructor
- The `self` keyword
- Creating Attributes (Instance Variables)
- Creating Methods (Instance Functions)
- `None` keyword
- Creating and using Objects (Instantiation)

## Day 17: Patient and Triage Classes

- OOP Architecture: Separation of Concerns
- Model (Data Class, e.g., `Patient` )
- Logic/Brain Class (e.g., `TriageLogic` )
- Main Controller file ( `main.py` )
- Importing classes from other files
- Using objects as attributes (e.g., `self.current_patient` )

## Day 18: Plot a Patient's Fever Chart

- The `turtle` module
- Coordinate Systems (X, Y axes)
- Turtle attributes and methods ( `.penup()` , `.pendown()` , `.goto()` , `.setworldcoordinates()` )
- Tuples (for RGB color data)
- Basics of Data Visualization (plotting data from a list)

## Day 19: A Simple Reaction Time Game

- OOP: Multiple, independent instances of objects
- Event-Driven Programming
- Event Listeners ( `.onclick()` )

- Higher-Order Functions (passing a function as an argument)
- Callback Functions (distinguishing `my_func` from `my_func()` )
- The `time` module: `time.time()`

**Day 20: Cell Mitosis Simulator (Part 1)**

- Composite Objects (objects built from other objects)
- Manual Game Loops ( `while True:` )
- Controlling Animation Speed ( `time.sleep()` )
- Controlling Screen Refreshes ( `screen.tracer(0)` , `screen.update()` )
- OOP Abstraction (hiding logic inside a class)

**Day 21: Specialized Medical Record Classes**

- Class Inheritance ( `class Child(Parent):` )
- The `super()` function
- Overriding and extending parent methods
- List Slicing (e.g., `my_list[1:]` , `my_list[-3:]` )
- The "is-a" vs. "has-a" relationship (Inheritance vs. Composition)

**Day 22: Hospital Simulation Components**

- OOP System Design: Object Interaction
- Mediator Class (e.g., `AppointmentScheduler` )
- Loose Coupling (objects not depending on each other's internal logic)
- Boolean attributes as flags ( `is_waiting` , `is_free` )

**Day 23: A "Virus Spreader" Simulation**

- The "Manager" or "Factory" OOP Pattern
- Object Lifecycle Management (Creating, Updating, and Deleting objects)
- Collision Detection (using `.distance()` )
- Managing lists of objects

**Day 24: Patient Report Generator**

- File I/O: Reading files ( `with open(...)` , `.read()` , `.readlines()` )
- File I/O: Writing files ( `mode='w'` )
- File I/O: Appending to files ( `mode='a'` )
- String Methods: `.replace()` , `.strip()`
- File Paths: Relative vs. Absolute
- `os` and `pathlib` modules (for cross-platform path building)

# Part 2: Data Science & Advanced Topics (Days 25-40)

### Day 25: Analyze Patient Data

- The `pandas` library
- `pd.read_csv()`
- DataFrames and Series
- Selecting Columns: `df['column_name']`
- Filtering Rows (Boolean Masking): `df[df['age'] > 50]`
- Descriptive Statistics: `.mean()` , `.max()` , `.idxmax()` , `.value_counts()`
- `df.info()` and `df.describe()` (Beyond Basics)

### Day 26: Medical Abbreviation Mapper

- List Comprehension: `[new_item for item in list if condition]`
- Dictionary Comprehension: `{new_key: new_value for item in list}`
- Looping through nested lists

### Day 27: Simple Medical Converter GUI

- `tkinter` GUI framework
- Widgets: `Window` , `Label` , `Entry` , `Button`
- Layout Managers: `.grid()` (preferred) vs. `.pack()` and `.place()`
- Widget Methods: `.get()` (from Entry) and `.config()` (for Label)
- Callback functions for buttons
- `*args` (unlimited positional arguments)
- `**kwargs` (unlimited keyword arguments)

### Day 28: "Take Your Pills" Reminder App

- Event-Driven Programming in Tkinter
- Non-Blocking Delays: `window.after()`
- Blocking Delays: `time.sleep()` (and why it's bad for GUIs)
- Dynamic UI Updates
- Canceling `after()` jobs: `window.after_cancel()`
- `tkinter.messagebox`

### Day 29: Medical Abbreviation Manager

- Integrating UI, Logic, and File I/O
- File I/O: Appending ( `mode='a'` )

- `tkinter.messagebox` ( `.showinfo` , `.showwarning` , `.askokcancel` )
- Clearing `Entry` widgets
- User Input Validation (checking for empty strings)

### Day 30: Refactor the Abbreviation Manager

- Exception Handling: `try` , `except` , `else` , `finally`
- Handling Specific Exceptions: `FileNotFoundError` , `KeyError`
- The `json` Module: `json.load()` , `json.dump()` , `json.update()`
- Refactoring code from `.txt` to `.json`

### Day 31: Medical Terminology Flashcard App

- `pandas.to_dict(orient='records')`
- `random.choice()`
- Separation of Logic (the "brain") and Presentation (the "face")
- Reading and writing CSV files to manage state
- `window.after()` for timed state changes

### Day 32: Medication Refill Reminder

- The `datetime` module: `datetime.now()` , `.day` , `.month` , `.weekday()`
- The `smtplib` module
- Connecting to SMTP servers (Gmail, etc.)
- `connection.starttls()`
- Security: App Passwords (vs. regular passwords)
- Security: Environment Variables (Beyond Basics)

### Day 33: OpenFDA Drug Recall Checker

- APIs (Application Programming Interfaces)
- The `requests` module: `requests.get()`
- API Endpoints
- Parsing JSON responses: `response.json()`
- API Status Codes (200, 404, etc.)
- Error Handling: `response.raise_for_status()`

### Day 34: A Medical Quiz GUI

- API Parameters: `params` dictionary in `requests.get()`
- Python Type Hinting: `variable: str` , `def func(arg: int) -> bool:`
- The `html` module: `html.unescape()`

- Connecting an API to a GUI application
- OOP: Refactoring logic into a `QuizBrain` class

**Day 35: "Sunscreen Alert" App**

- API Authentication: API Keys
- Environment Variables: `os.environ.get()`
- Security: Hiding secret keys from code
- The `.gitignore` file
- Parsing complex nested JSON responses

**Day 36: Drug Information & News Alert**

- Chained Logic: "If-Then" workflows
- Conditional API calls (to save resources/money)
- Mocking data (using dictionaries and functions) for testing
- List Slicing (e.g., `news_list[:3]` )

**Day 37: Create a Mock Patient on a Test API**

- HTTP Verbs: `GET` , `POST` , `PUT` , `PATCH` , `DELETE`
- RESTful API Principles
- `requests.post()` : Sending data with the `json=` parameter
- API Headers
- Reading API documentation (e.g., `reqres.in` )
- Checking `response.status_code` (e.g., `201 Created` )

**Day 38: Patient Feedback Analyzer**

- Natural Language Processing (NLP) (Conceptual)
- Integrating multiple specialized APIs (Service-oriented architecture)
- `requests.post()` with authentication headers
- Authentication: Bearer Tokens
- Creating a structured log from unstructured data

**Day 39 & 40: Architect Your Symptom2Specialist Bot**

- System Architecture Design
- OOP: Single Responsibility Principle
- OOP: Abstraction (defining class "contracts")
- Creating placeholder "shell" classes and methods ( `pass` keyword)
- Writing "Project Manager" logic ( `main.py` ) to coordinate components

# Part 3: Web Development & Automation (Days 41-60)

### Day 41 & 42: A Static Patient Profile Page

- HTML: Boilerplate ( `<!DOCTYPE html>` , `<html>` , `<head>` , `<body>` )
- HTML: Content Tags ( `<h1>-<h6>` , `<p>` , `<a>` , `<img>` )
- HTML: List Tags ( `<ul>` , `<ol>` , `<li>` )
- HTML: Table Tags ( `<table>` , `<tr>` , `<th>` , `<td>` )
- HTML: `div` and `span`
- Semantic HTML: `<nav>` , `<main>` , `<section>` , `<article>` , `<footer>`

### Day 43 & 44: Style the Patient Profile Page

- CSS: The Box Model (Margin, Border, Padding, Content)
- CSS Selectors: Tag, Class, ID
- CSS: External, Internal, and Inline styles
- Linking an external stylesheet
- Common CSS Properties: `color` , `background-color` , `font-family` , `font-size` , `text-align` , `border` , `border-radius` , `margin` , `padding`
- Using Google Fonts

### Day 45: Scrape a Medical Wikipedia Page

- Web Scraping Concepts
- The `requests` library
- The `BeautifulSoup` library
- Parsing HTML: `BeautifulSoup(html_content, "html.parser")`
- Finding elements: `soup.find()` , `soup.find_all()`
- CSS Selectors: `soup.select()`
- Extracting data: `.getText()` and `.get("attribute_name")`
- Using Browser "Inspect" tool

### Day 46: Search PubMed with Selenium

- The `selenium` library
- `WebDriver` (e.g., `chromedriver` )
- Automating a browser: `driver.get(URL)`
- Dynamic vs. Static Websites
- Finding elements: `driver.find_element(By.NAME, ...)`
- Interacting with elements: `.send_keys()`
- Using special keys: `Keys.ENTER`

- Closing the browser: `driver.quit()`

### Day 47-50: Practo.com Pop-up Handler

- Advanced Selenium: `WebDriverWait`
- Advanced Selenium: `expected_conditions` (e.g., `element_to_be_clickable` )
- Handling `TimeoutException` with `try...except`
- Handling Pop-ups / Modals
- Interacting with complex forms (logging in, filling fields)

### Day 51: "Website Down" Alerter

- System Monitoring Logic: "Sense -> Decide -> Act" loop
- Combining `selenium` with `try...except TimeoutException` as a "sensor"
- Abstracting automation into classes (Beyond Basics)
- Storing selectors in a separate config file (Beyond Basics)

### Day 52: Scroll a Medical Forum Thread

- Handling Modals / Pop-ups (follower list example)
- Infinite Scrolling
- Executing JavaScript: `driver.execute_script()`
- Scrolling to bottom: `window.scrollTo(0, document.body.scrollHeight)`
- Detecting end of scroll
- Randomizing `time.sleep()` to mimic human behavior (Beyond Basics)

### Day 53: Scrape and Log Medical News

- Hybrid Scraping: `requests` + `BeautifulSoup` for speed, `selenium` for interaction
- Mimicking a browser: Setting `User-Agent` headers in `requests`
- Automating Google Forms
- Looping through scraped data to fill a form

### Day 54 & 55: Simple Symptom Checker API

- Web Frameworks: `Flask`
- Decorators: `@app.route()`
- Running a local web server
- Returning HTML from Python
- Dynamic URLs: `@app.route('/<string:symptom>')`
- URL Variable Converters ( `<int:id>` , `<string:name>` )

### Day 56 & 57: Dynamic Patient List Page

- Web Templates: Separating Logic from Presentation
- `render_template()` function
- `Jinja` Templating Engine
- Jinja Syntax: `{{ variable }}`
- Jinja Logic: `{% for item in list %}` and `{% if condition %}`
- Template Inheritance: `{% extends "base.html" %}` and `{% block content %}`
- Using the `static` folder and `url_for()`

### Day 58, 59 & 60: A Simple Patient Intake Form

- CSS Frameworks: `Bootstrap` (adding via CDN)
- Using Bootstrap classes ( `container` , `btn` , `form-control` )
- HTML Forms: `<form action="..." method="POST">`
- Input `name` attributes
- Handling `POST` requests in Flask: `methods=["GET", "POST"]`
- Accessing form data: `request.form.get("name")`
- `redirect()` function

## Part 4: Full-Stack & Data Science (Days 61-80)

### Day 61: A Secure Patient Intake Form

- `Flask-WTF` library
- CSRF Protection: `form.hidden_tag()`
- Defining forms as Python classes
- Form Fields: `StringField` , `IntegerField` , `SubmitField`
- Form Validators: `DataRequired()` , `Email()` , `Length()`
- Validating forms: `form.validate_on_submit()`
- Accessing data: `form.field_name.data`

### Day 62: Medical Clinic Locator

- Application Integration: `Flask` + `Flask-Bootstrap` + `Flask-WTF` + `csv`
- Python `csv` module: `csv.reader` , `csv.writer`
- Full CRUD with a CSV file (Read, Add)
- Redirecting after form submission

### Day 63: Refactor the Clinic Locator

- Databases: SQLite
- ORM (Object Relational Mapper)
- `Flask-SQLAlchemy` library
- Defining Models: `class MyTable(db.Model):`
- Defining Columns: `db.Column()` , `db.Integer` , `db.String` , `primary_key=True`
- Creating the database: `db.create_all()`
- Creating records: `db.session.add()` & `db.session.commit()`
- Reading records: `Model.query.all()` , `Model.query.get()`

### Day 64, 66-67: Build a Medication Tracker API

- CRUD Operations: Create, Read, Update, Delete
- Update: `Model.query.get()` , update fields, `db.session.commit()`
- Delete: `db.session.delete()` , `db.session.commit()`
- RESTful API Design Principles
- `jsonify()` for returning JSON responses
- HTTP Methods: `GET` , `POST` , `PATCH` , `DELETE`
- Testing APIs with Postman or Insomnia

### Day 65: Redesign the Medication Tracker

- UI/UX Principles
- Color Theory & Palettes (e.g., `Coolors.co` )
- Typography & Readability (e.g., `Google Fonts` )
- Layout: Whitespace, Grids, "Card" design
- Favicons

### Day 68-70: A Full-Stack Medication Tracker

- Authentication: Password Hashing ( `werkzeug.security` )
- `Flask-Login` : `LoginManager` , `UserMixin` , `login_user` , `logout_user` , `@login_required`
- Database Relationships: One-to-Many
- `db.ForeignKey`
- `db.relationship`
- Filtering queries: `.filter_by()`
- Deployment Prep: `requirements.txt` ( `pip freeze > ...` )
- Deployment Prep: `.gitignore` file
- Deployment Prep: `Procfile` and `gunicorn`
- Deployment Prep: Environment variables for database URI

### Day 71: Explore Patient Demographics

- `pandas.read_csv()`
- `df.head()`
- `df.columns`
- Selecting columns ( `df['col']` ) and rows ( `df.loc[]` )
- Aggregations: `.mean()` , `.max()` , `.idxmax()`

### Day 72: Clean Lab Results

- Identifying Missing Data: `NaN` values, `df.isna().sum()`
- Handling Missing Data: `df.dropna()`
- Handling Missing Data: `df.fillna()`
- Grouping Data: `df.groupby('col')`
- Chaining methods: `df.groupby('col').mean()`

### Day 73: Consolidate Patient Visit Data

- Merging DataFrames: `pd.merge(df1, df2, on='common_key')`
- SQL JOIN types (Inner, Outer, Left, Right)
- Pivoting DataFrames: `df.pivot_table(index=..., columns=..., values=..., aggfunc=...)`

### Day 74 & 75: Visualize Patient Vitals

- Data Visualization (EDA)
- `matplotlib.pyplot` : `plt.scatter()` , `plt.title()` , `plt.xlabel()` , `plt.ylabel()` , `plt.show()`
- `seaborn` : `sns.regplot()` (for linear regression lines)
- `plotly.express` : `px.scatter()` (for interactive plots)
- Choosing the right plot (scatter, line, bar, histogram)

### Day 76-79: Predict Diabetes Risk

- `numpy` library: Arrays and Vectorization
- `scikit-learn` library
- Supervised Learning: Regression vs. Classification
- Features (X) vs. Target (y)
- `train_test_split()`
- Model 1: `LogisticRegression`
- Model 2: `RandomForestClassifier`
- Training: `model.fit(X_train, y_train)`
- Predicting: `model.predict(X_test)`

- Evaluation: `accuracy_score()` , `confusion_matrix()`

**Day 80: Heart Disease Risk Analysis**

- End-to-End Data Science Workflow
- i. Asking Questions
- ii. Data Wrangling (Cleaning)
- iii. Exploratory Data Analysis (EDA) & Visualization
- iv. Modeling & Prediction
- Presenting results (Jupyter Notebooks)

## Part 5: Portfolio Projects (Days 81-100)

**Day 81 & 82: A Medical Abbreviation Translator**

- Dictionary as a mapping tool
- `tkinter` UI: `Entry` , `Button` , `Label`
- Exception Handling: `try...except KeyError`
- String methods: `.upper()`

**Day 83 & 84: Command-Line Symptom Diagnosis Game**

- State Management (game board, current player)
- Algorithmic Logic: Checking win conditions (rows, cols, diagonals)
- Pattern Recognition (matching board state to "diagnoses")
- Separating Game Logic from UI (Beyond Basics)

**Day 85 & 86: "Confidential" Lab Report Stamper**

- `Pillow` (PIL) library: `Image.open()` , `Image.save()`
- `ImageDraw.Draw()`
- `ImageFont.truetype()`
- `draw.text()`
- `tkinter.filedialog.askopenfilename()`
- RGBA Colors (for transparency)
- Advanced: Image rotation and merging (Beyond Basics)

**Day 87 & 88: A Public Health Data Viewer**

- Model-View-Controller (MVC) Pattern
- Python `csv` module: `csv.reader`
- `Flask` and `render_template`

- `Jinja` `for` loops
- `Bootstrap` for styling tables

**Day 89 & 90: A "Smart" Medical Note Pad**

- `tkinter.Text` widget
- `Text` widget tagging ( `.tag_add` , `.tag_config` )
- Event Binding: `<KeyPress>`
- Event Loop: `window.after_cancel()`
- Event Loop: `window.after()`
- The "Debouncing" UI Pattern

**Day 91 & 92: X-Ray Feature Extractor**

- Image Feature Extraction (Conceptual)
- `Pillow` : `Image.open()` , `.size` , `.crop()` , `.getdata()`
- Image coordinate systems (bounding boxes)
- Grayscale images and pixel intensity
- `numpy` (Beyond Basics, for faster array math)

**Day 93 & 94: "Red Alert" Heart Rate Monitor Bot**

- Screen Scraping: `Pillow.ImageGrab.grab()`
- Pixel-level analysis (getting average color)
- "See -> Analyze -> React" loop
- `pyautogui` or `selenium` for programmatic clicking (Beyond Basics)
- Building "brittle" bots and their limitations

**Day 95 & 96: PubMed Scraper API**

- "Scraper-as-a-Service" Architecture
- Wrapping a scraper ( `requests + BeautifulSoup` ) in a `Flask` server
- `Flask` dynamic routes ( `@app.route('/api/...')` )
- `Flask.jsonify`
- Caching with `Redis` (Beyond Basics)

**Day 97: Deploy Your Patient List App**

- Deployment Concepts
- `pip freeze > requirements.txt`
- Cloud Hosting Platforms (e.g., PythonAnywhere, Render)
- WSGI Servers ( `gunicorn` )

- Web App Configuration (WSGI file)
- Using Environment Variables in production

**Day 98-100: Hospital Readmission Predictor**

- End-to-End ML Workflow
- Data Cleaning & Wrangling (Handling `NaN`, `pd.read_csv`)
- Feature Engineering: `pd.get_dummies()` (One-Hot Encoding)
- Feature Selection (X) vs. Target (y)
- `scikit-learn`: `train_test_split()`
- `scikit-learn`: `LogisticRegression` (for classification)
- Model Evaluation: `accuracy_score()`, `confusion_matrix()`
- Interpreting results (False Positives vs. False Negatives)