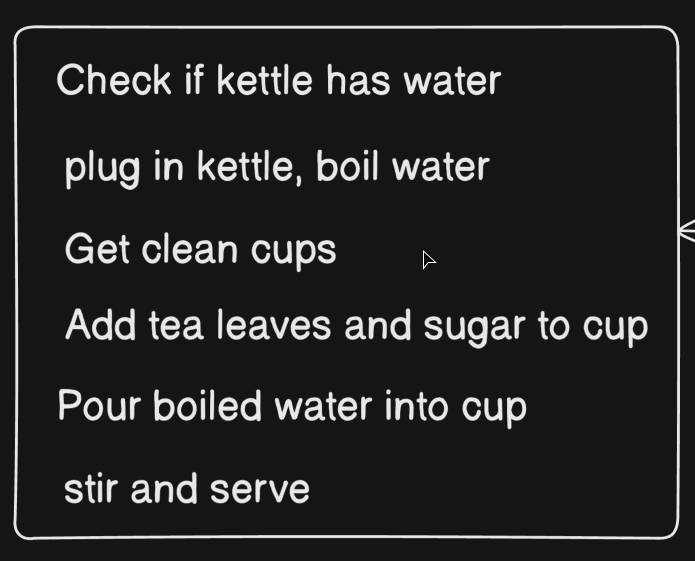
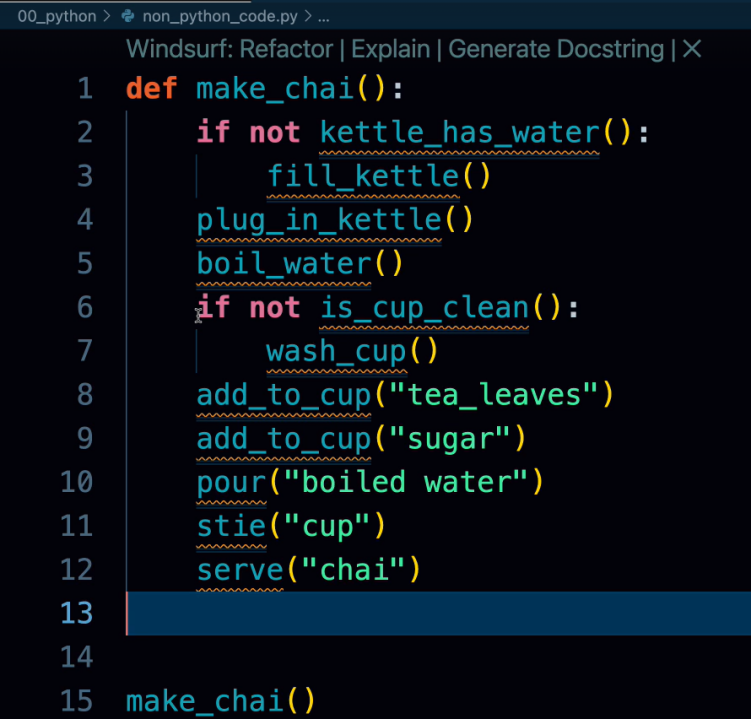
[Full-Stack AI with Python: LLMs, RAG, Agents & LangGraph](https://www.udemy.com/course/full-stack-ai-with-python/)

1. **Section 1: Introduction**
   1. **1. Installation of Tools (VSCode and Python)**
      1. IDE: Download VSCode.
      2. Install python latest version
         1. `python --version`
         2. In my cmd: Python 3.12.1
         3. In my WSL: Python 3.10.12
         4. Know the basic of python like variable, functions, math operations, classes
            1. `python` in terminal -> Open REPL
      3. Will use OpenAI, Gemini and other LLM’s. be aware of Chatgpt and have am account on openai. Will wak through on how to add credits in openai account.
   2. **2. VS Code Setup (Extensions and Themes)**
      1. Pylance, Ayu-miraj(theme) vscode extention
   3. **3. SRC code received**
2. **Section 2: Introduction to Coding world with python**
   1. **4. Meet your Instructor – Hitesh**
      1. Hitesh Intro
   2. **5. What is Programming..?**
      1. Install VS code
      2. <https://app.eraser.io>, <https://tldraw.com>, <https://excalidraw.com> -> drawing tool
      3. Programming –
         1. It’s about giving instructions to computer, especially the instruction that computer do understand.
         2. Computers can’t think on their own, including AI (just patterns & word completions).
         3. Requires precise instructions. Eg: making chai, we need to give set of instructions to make chai.
         4. Key Points on Coding Difficulty
            1. Is coding easy? → No and Yes.

Writing basic code: easy to start.

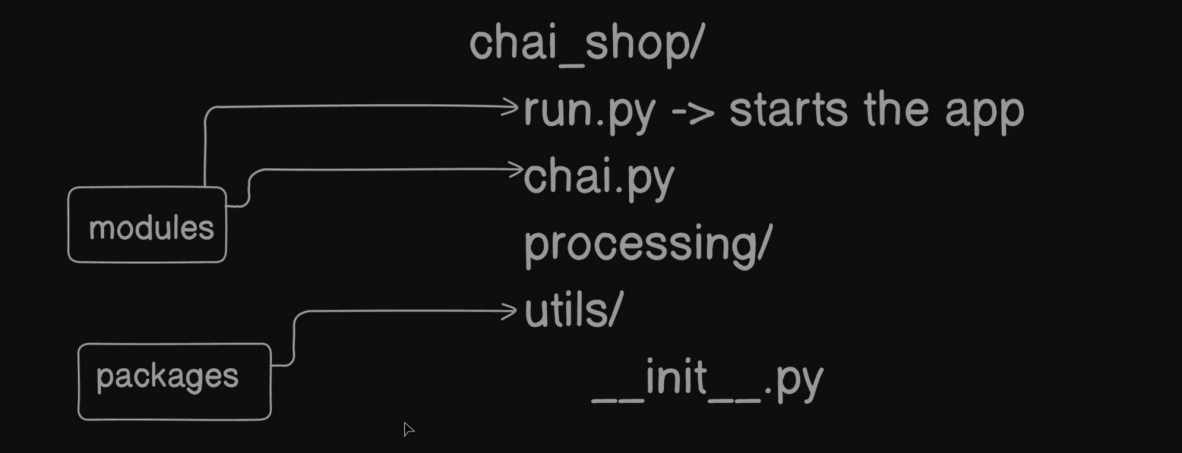
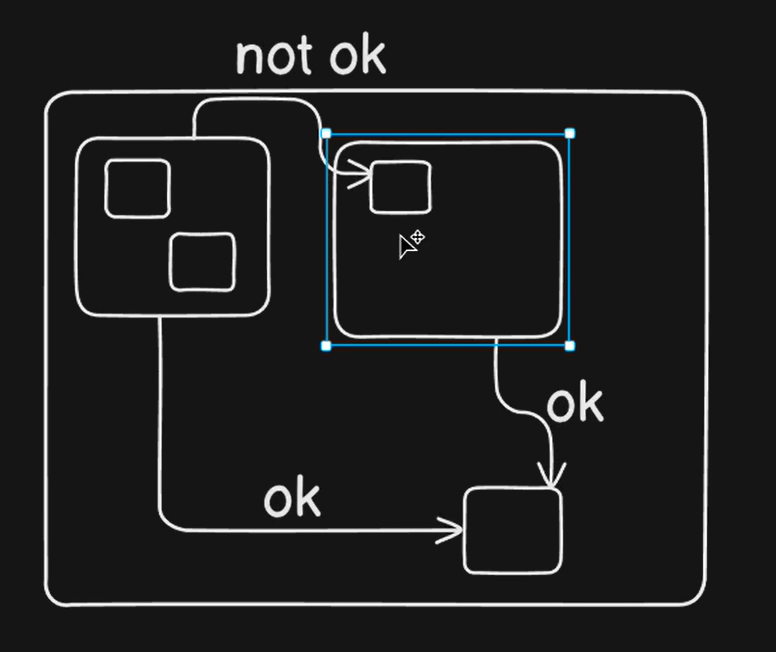
Becoming good programmer: takes months to years of effort.

* + - * 1. Coding is about problem breakdown and thinking process, not just writing syntax.
        2. Simple programs can be learned in a few months (Python & JavaScript more beginner-friendly).
  1. **6. Convert that into Python Code**
     1. Write recipes in code. Main code in each separate fx.
        1. Write this box items in python code.
           1. 
        2. Python code
           1. 
  2. **7. A Real World Python Code Intro**
     1. Python program terms
        1. Method/Functions -> Reusable piece of code in a fx.
        2. Class: contains methods/fxs
        3. Object: when class is initialized
        4. Properties: attributes
     2. Eg program:
        1. not every class in Python needs to have an \_\_init\_\_ dunder method.
        2. In class we have to pass self as an argument in every fx.
        3. Code eg: refer [my\_code\00\_python\1\_non\_python\_shop.py](my_code/00_python/1_non_python_shop.py)
  3. **8. Why to use Python?**
     1. Easy lang, Readable
     2. Portable: same program runs on windows, mac, linux
     3. Productivity: python is slower than cpp/java but same task can be done very easily.
     4. Extensive support of free libraries.
     5. Multi-use lang: make fullstack apps, ML/AI/datascience, Automation.
     6. Chai level happiness.
  4. **9./10. Writing first Python code on MAC/Windows**
     1. You can use python REPL or .py files to write python code.
     2. `python3 --version` -> in terminal to see the python version
     3. <https://www.warp.dev/> -> AI CLI
  5. **11. Get everything in Virtual Environment**
     1. Virtual ENV: safeguarding your apps from other python versions and its libraries version. And don’t cluter your OS.
     2. How to Create a Virtual Environment (Traditional Way with venv)
        1. Create a Project Directory
           1. `mkdir project`
           2. `Cd project`
        2. Create the Virtual Environment:
           1. `python3 -m venv .venv`
        3. Activate the Virtual Environment:
           1. On windows: `venv\Scripts\activate`
           2. On linux: `source venv/bin/activate`
        4. Deactivate the Environment:
           1. `deactivate`
        5. Installing Third-Party Packages
           1. Once the environment is activated, any package you install will be local to that environment.

`pip install flask`

* + - * 1. requirements.txt

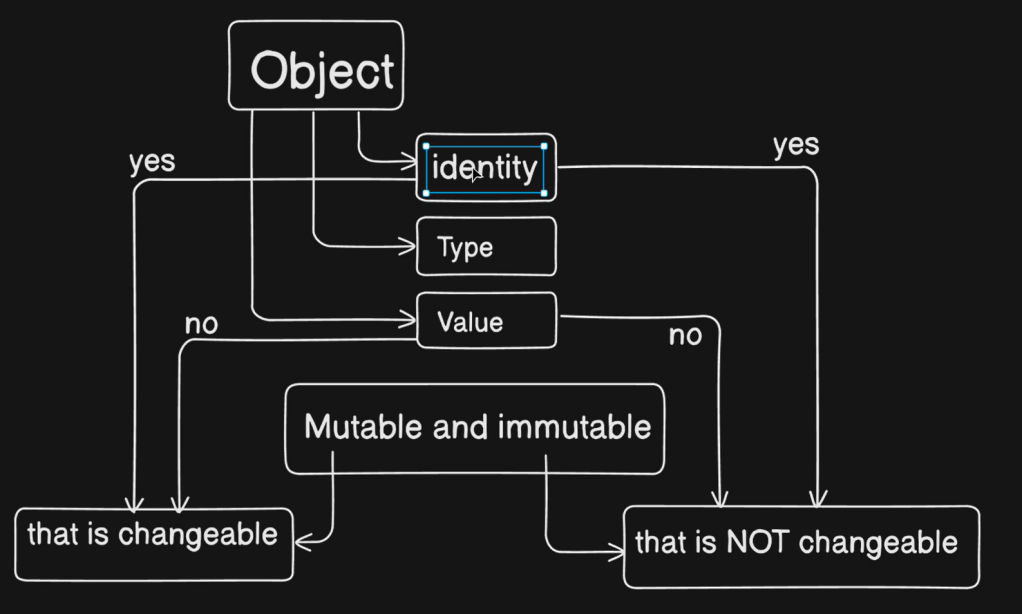
`pip install -r requirements.txt`

* + - * 1. You can specify exact versions of python libraries/modules for reproducibility.
    1. *Best Practices*
       1. Always create a new virtual environment for each project.
       2. Never install packages globally for development projects.
       3. Use requirements.txt to specify dependencies.
       4. Do not commit the venv folder to version control systems (add it to .gitignore).
       5. Share source code and requirements.txt so others can recreate your environment.
       6. Only source code and requirements.txt are shared.
       7. Recipients create a new virtual environment and install dependencies using the requirements.txt file.
    2. Modern Alternative: UV Tool
       1. uv is a new tool that simplifies and speeds up virtual environment management compared to venv.
       2. Offers superior dependency solving and faster installs.
  1. **12. Organize Python Code like a Pro**
     1. Package(\_\_init\_\_.py) vs modules vs regular folder
        1. 
     2. Namespace and scope concepts using the helpful "house analogy"
        1. 
        2. In a city every house(class) can access public as well as its own house content but one house can and should not access the other house content.
  2. **13. PEP8 and Zen of python**
     1. View PEP8 doc
     2. Zen of python: `import this`

1. **Section 3: Data types in python**
   1. **14. Objects - Mutable and Immutable in Python**
      1. When learning a programming language, two key aspects need to be understood:
         1. Types of Data (Data Types)
            1. Numbers (integers, decimals, imaginary numbers)
            2. Strings (names, text)
            3. Collections (sets, lists, etc.)
         2. How to Manipulate Data
            1. Operations on numbers (addition, multiplication)
            2. Operations on strings (case conversion, email validation, etc.)
      2. Python Objects
         1. In Python, everything is an object.
         2. Every object has three primary attributes:
            1. Identity (a unique id of the object in memory)
            2. Type (defines the nature of the object, e.g., int, str, set)
            3. Value (the actual data stored, e.g., 2, "Hitesh")
         3. Objects can have an empty value as well.
      3. Mutability and Immutability
         1. Mutable = Changeable (the object itself can be modified).
         2. Immutable = Not changeable (the object cannot be modified, only new objects can be created).
         3. Key Rule:
            1. Never check *mutability using values, always use identity* (memory id) to determine.

If the identity remains the same after modification → object is mutable.

If the identity changes when its "value" changes → object is immutable.

* + - * 1. 
        2. Examples

Immutable Example (Numbers)

Numbers in Python are immutable.

Example:

Initially: sugar\_amount = 2

Later reassigned: sugar\_amount = 12

The value seems to change, but in reality*, reference is changes for that variable*:

Python created a new object (12).

Variable reference was pointed from 2’s memory to 12’s memory.

Identity (id) of 2 and 12 are different, proving immutability.([my\_code\02\_datatypes\Chapter\_1.py](my_code/02_datatypes/Chapter_1.py))

2. Mutable Example (Set)

Sets are mutable objects.

Example:

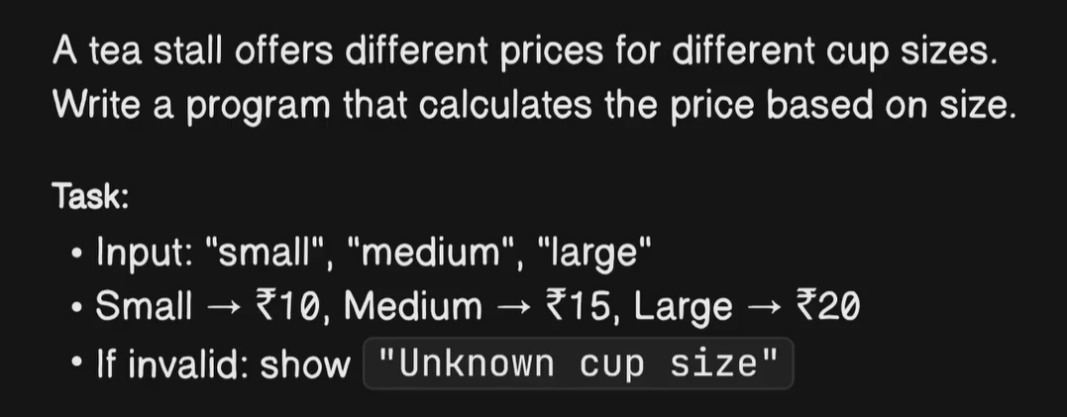
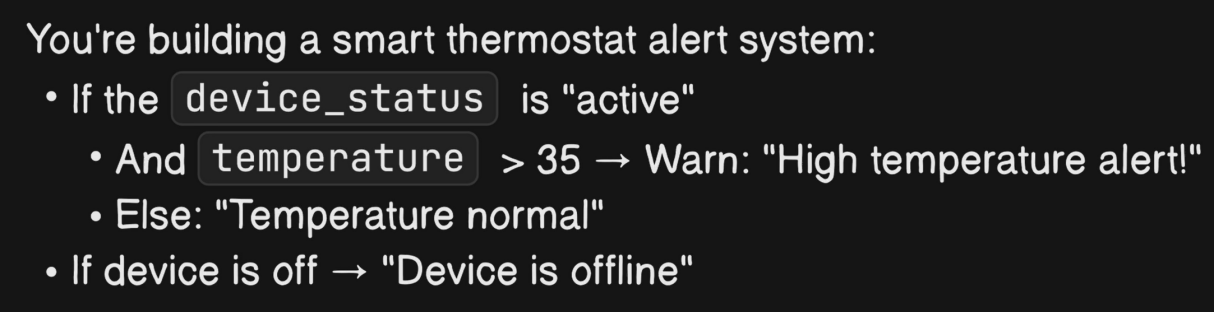
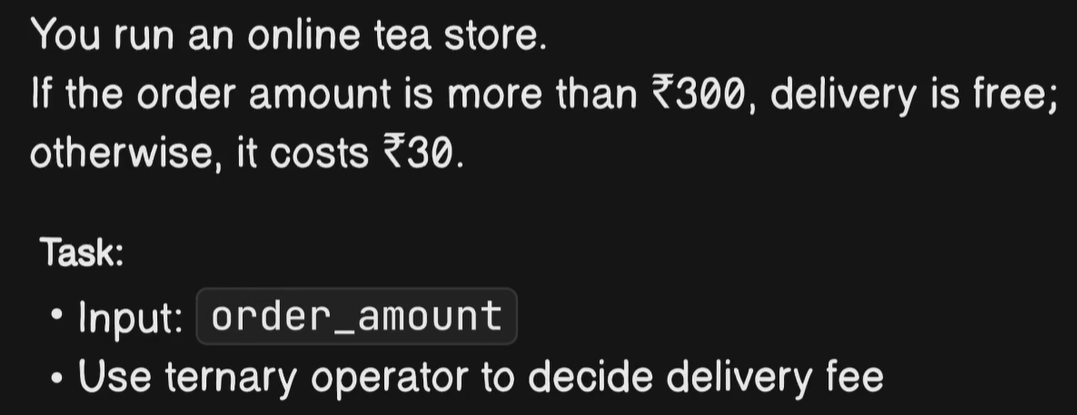
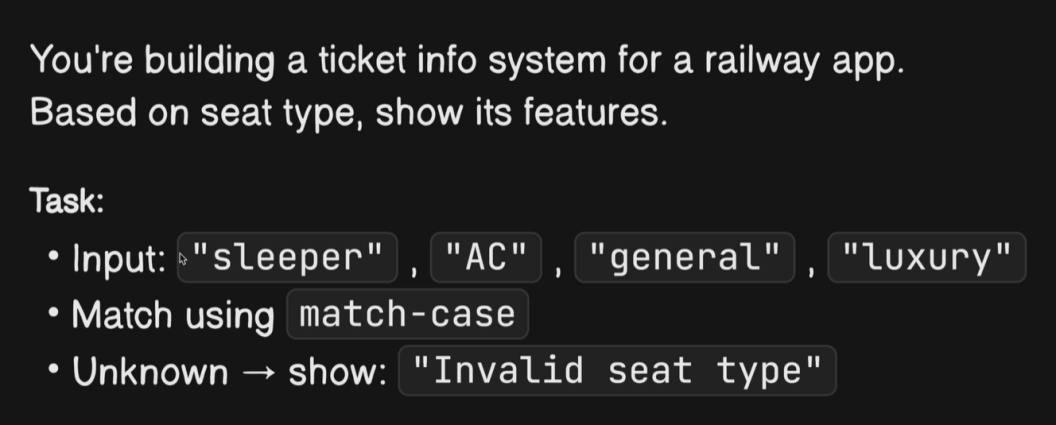
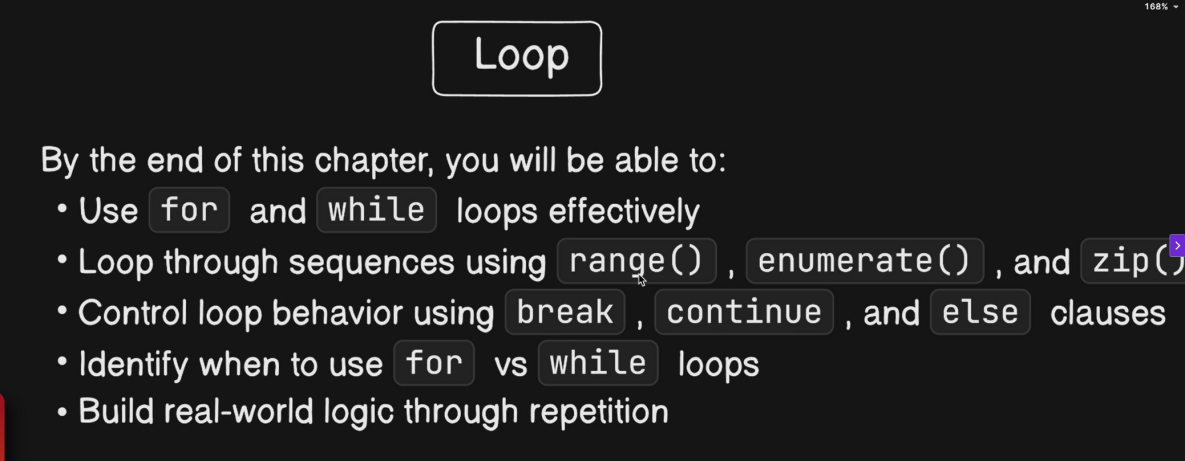
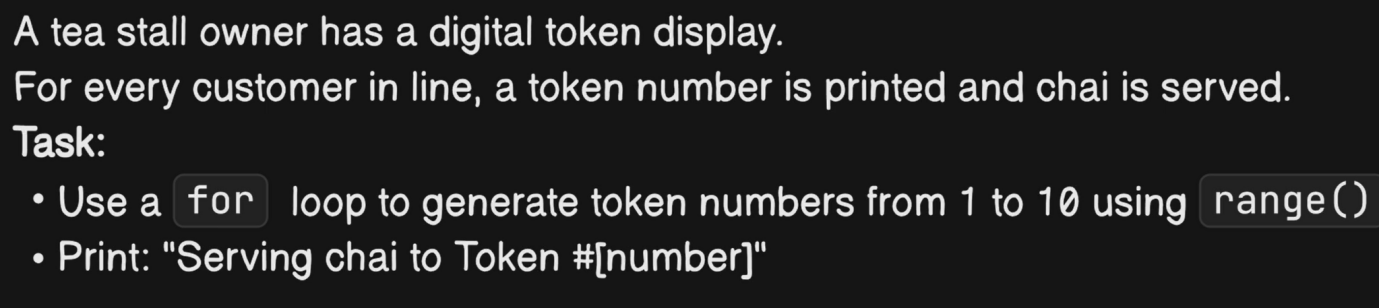
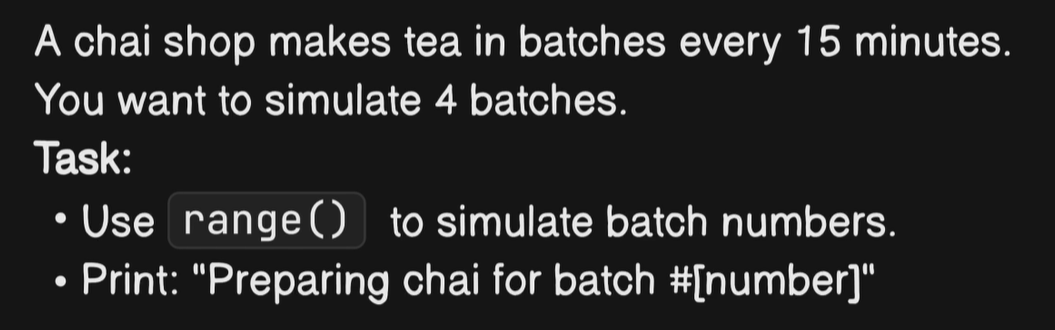
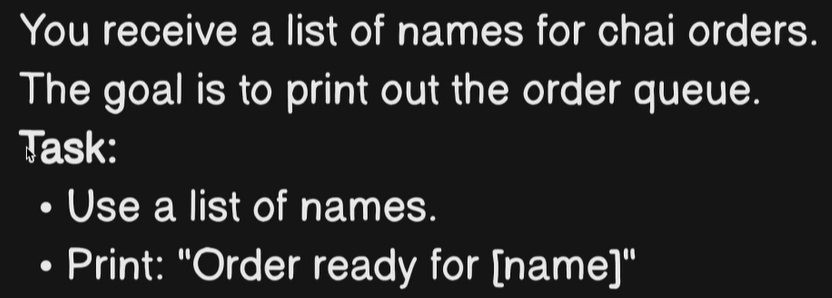
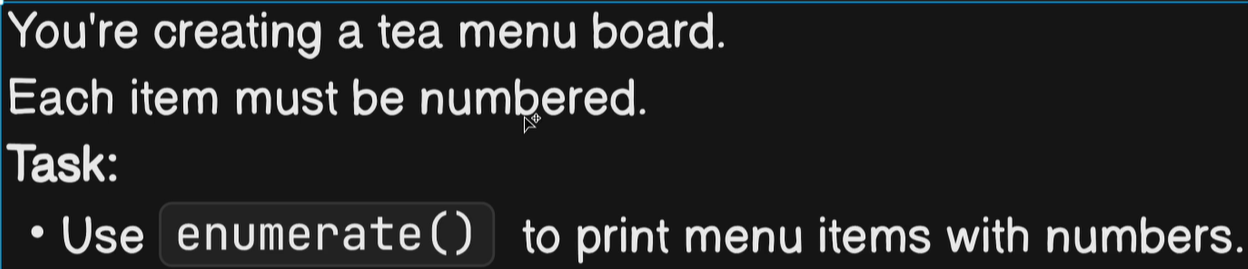
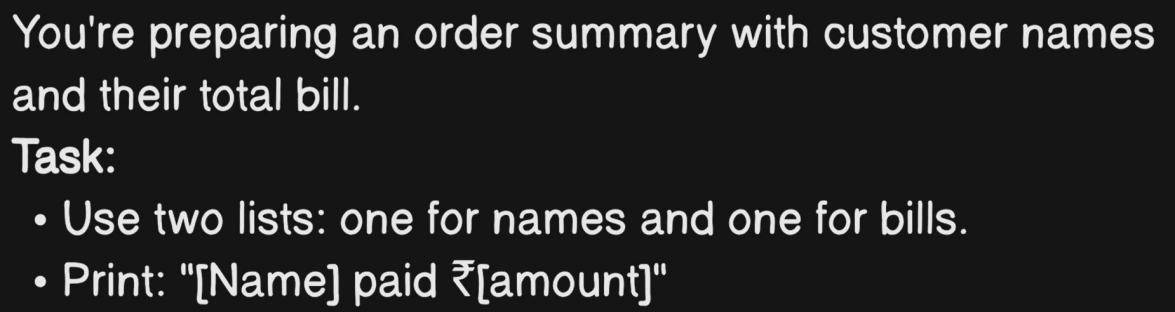
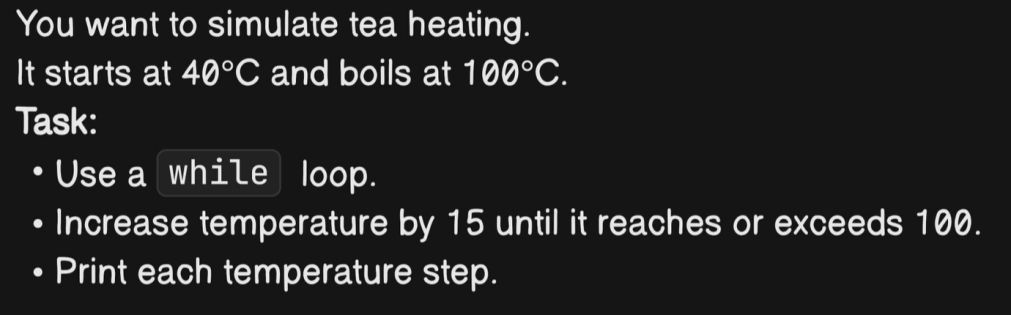
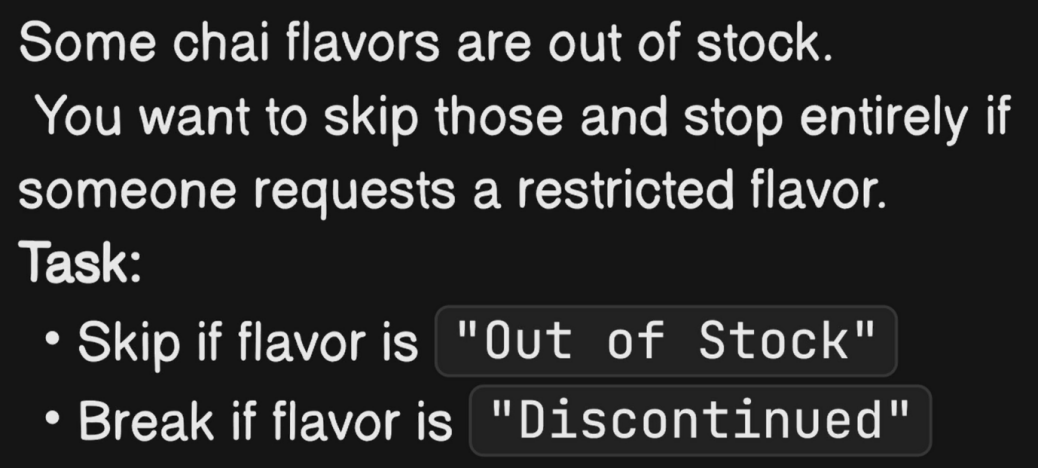
Start: spice\_mix = set() → Empty set created.Code: [my\_code\02\_datatypes\Chapter\_2.py](my_code/02_datatypes/Chapter_2.py)

Add items: spice\_mix.add("ginger"), spice\_mix.add("cardamom")

Identity of spice\_mix stays the same, but its contents change in memory.

This shows that the same object can be changed without creating a new one.

* + 1. Key Takeaways
       1. Objects = identity + type + value.
       2. Immutables (e.g., numbers, strings, tuples):
          1. Cannot be modified.
          2. Reassignments create new objects.
       3. Mutables (e.g., sets, lists, dictionaries):
          1. Can be modified in place.
          2. Identity remains the same even when contents change.
       4. Always verify mutability via identity, not by checking values.
  1. **15. Numbers, Booleans and Operators in Depth in Python**
     1. Types of Numbers in Python
        1. Integers (int) -> 1, 2
        2. Booleans (bool) -> True, False
        3. Real Numbers / Floating Point (float) -> 1.234
        4. Complex Numbers (complex) -> 1+ 2j
     2. Arithmetic Operations in Python([my\_code\02\_datatypes\Chapter\_3.py](my_code/02_datatypes/Chapter_3.py))
        1. Addition: a + b
        2. Subtraction: a - b
        3. Multiplication: a \* b
        4. Division (Float Division): a / b → Produces decimal (e.g., 7 / 4 = 1.75)
        5. Floor Division: a // b → Ignores decimal part (e.g., 7 // 4 = 1)
        6. Modulo: a % b → Returns remainder (e.g., 10 % 3 = 1)
        7. Exponentiation: a \*\* b → Raises to power (e.g., 2 \*\* 3 = 8)
        8. Readability: var = 1\_000\_000 -> 1000000
     3. Logical Operators
        1. And, or, not
     4. Floating Point and Precision
        1. Precision errors may occur in floating numbers.
        2. Python supports scientific computing with fractions, decimal, and system-level float precision data.
  2. **16. String - Index, Slice and Encoding**
     1. Code file: [my\_code\02\_datatypes\Chapter\_6.py](my_code/02_datatypes/Chapter_6.py)
     2. String are immutable data type(cannot be changed in place; new reference is created in memory when modified).
     3. A string is a sequence of characters enclosed in quotes
     4. Slicing:
        1. Syntax: string[start:end:step]
        2. start: index to begin (inclusive).
        3. end: index to stop (exclusive).
        4. step: interval (positive → forward, negative → reverse).
     5. Encoding and Decoding Strings
        1. Useful when working with non-English characters (Hindi, Tamil, Japanese, Spanish accents, etc).
        2. Encode converts string to a specified format (typically UTF-8).
        3. Decode converts encoded string back to readable form.
     6. Strings have many built-in methods (like .upper(), .lower(), .count(), .capitalize(), etc.) to manipulate text.
  3. **17. Tuples and Membership Testing**
     1. What are Tuples?
        1. Code: [my\_code\02\_datatypes\Chapter\_7.py](my_code/02_datatypes/Chapter_7.py)
        2. Core data types, Syntax: tuple1 = (1,)
        3. Immutable → contents cannot be changed after creation.
           1. Cannot be modified (no adding, removing, or reassigning elements).
        4. Often used when you want fixed collections of items.
        5. Elements can be accessed, unpacked, and tested for membership(case sensistive).
        6. You can swap values without requiring a third variable.
  4. **18. Basics of List in Python**
     1. Immutable Data Types
        1. Once a memory reference is assigned, it cannot be changed.
        2. Examples studied so far: immutable types (e.g., numbers, tuples, strings).
        3. To change values, a new object has to be created (e.g., number 2 changed to 3).
        4. Immutable data types have a unique ID that never changes once created.
     2. Mutable Data Types
        1. Code: [my\_code\02\_datatypes\Chapter\_8.py](my_code/02_datatypes/Chapter_8.py)
        2. Can be changed after creation.
        3. for list data type, Lists allow mixing data types and reordering elements. *Methods* like append, insert, remove, pop, sort, reverse exist because lists are mutable.
        4. Max and Min algo on list.
  5. **19. Operator overloading and bytearray in python** 
     1. Operator Overloading
        1. Definition: When an operator (like +, -, \*) performs more than one task, depending on context.
        2. Code: [my\_code\02\_datatypes\Chapter\_8.py](my_code/02_datatypes/Chapter_8.py)
        3. In Python, + also support operator overloading.
        4. Python has an operator module for advanced list manipulations.
           1. itemgetter returns a callable object that retrieves specific items from data structures.
  6. **20. Set and frozenset in python**
     1. uniqueness — sets do not allow duplicate values.
        1. Unordered collection of unique items.
        2. Immutable values, but the set itself is mutable (you can add/remove items).
     2. Set operations:
        1. Intersection (&)
        2. Union (|)
        3. Difference (-)
        4. Membership test (in)
     3. Frozen Set
        1. Immutable version of a set.
        2. frozenset()
        3. Useful when you want a fixed, unchangeable set of unique elements.
        4. Behavior is the same as a normal set, except it cannot be modified.
     4. Code: [my\_code\02\_datatypes\Chapter\_9.py](my_code/02_datatypes/Chapter_9.py)
  7. **21. Dictionary in Python**
     1. Problem with Lists: Access values only with numeric indexes (0, 1, 2...).
        1. Solution: Dictionary provides named indexing (key-value pairs).
     2. Dictionary = key → value mapping.
     3. Order doesn’t guarantee — data is accessed via keys, not index.
     4. Dictionary is a mutable data type: can add, update, and delete key-value pairs.
     5. Keys must be unique and immutable (e.g., string, number, tuple).
     6. Values can be any data type (string, number, list, even another dictionary).
     7. Code: [my\_code\02\_datatypes\Chapter\_10.py](my_code/02_datatypes/Chapter_10.py)
     8. Dictionary operations behave similarly to some set operations (membership tests, unions, etc.).
  8. **22. Touch on Advance Data types like Collections**
     1. Advanced data types in Python are not built-in; they require third-party modules or special imports.
     2. Beginners don’t typically need them, but knowing they exist helps for later projects.
     3. Purpose: Provide additional functionality beyond Python’s core data structures.
     4. Modules:
        1. Datetime
        2. Time
        3. Calender
        4. Timedelta -> when diff between time
        5. Arrow
        6. Dateutils
        7. Python collections module
           1. collections provides specialized containers beyond lists, sets, and dictionaries.
           2. `from collections import namedtuple, deque, Counter, OrderedDict, defaultdict, ChainMap`
     5. Advanced data types provide more specialized tools but are built on top of basic Python structures.
     6. Beginners should focus on core data types first; advanced ones become essential as use cases grow more complex.
     7. Code: [my\_code\02\_datatypes\Chapter\_11.py](my_code/02_datatypes/Chapter_11.py)

1. **Section 4: Conditionals in python**
   1. **23. Kettle Boiling Story Project**
      1. Up to this point: learned about data types (numbers, strings, dates, time, etc.).
      2. Next: moving into processing data, applying logic and operations to data.
      3. Conditionals are used to make decisions in programs.
      4. A condition always evaluates to True or False.
      5. Shows how decisions branch based on Boolean answers.
      6. Code: [my\_code\03\_conditionals\mini\_story\_1.py](my_code/03_conditionals/mini_story_1.py)
   2. **24. Building a Snack System**
   3. 
      1. Code: [my\_code\03\_conditionals\snack\_suggestion\_2.py](my_code/03_conditionals/snack_suggestion_2.py)
   4. **25. Building a Chai Price Calculator**
      1. Code: [my\_code\03\_conditionals\chai\_price\_calculator\_3.py](my_code/03_conditionals/chai_price_calculator_3.py)
   5. **26. Building Smart Thermostat System**
      1. 
      2. [my\_code\03\_conditionals\smart\_thermostat\_4.py](my_code/03_conditionals/smart_thermostat_4.py)
   6. **27. Delivery Fees Waiver System** 
      1. 
      2. [my\_code\03\_conditionals\delivery\_fees\_wavier\_5.py](my_code/03_conditionals/delivery_fees_wavier_5.py)
   7. **28. Build a train seat information system**
      1. 
      2. [my\_code\03\_conditionals\train\_seat\_6.py](my_code/03_conditionals/train_seat_6.py)
2. **Section 5: Loops in python**
   1. **29. Introduction to Loops**
      1. Example: Displaying multiple pieces of data (like books retrieved from a database).
      2. Loops are a way to perform tasks repeatedly.
      3. 
      4. Range()-> [1,6) -> 1, 2, 3, 4, 5
   2. **30. Tea Token Dispenser**
      1. 
      2. [my\_code\04\_loops\1\_chai\_token\_queue.py](my_code/04_loops/1_chai_token_queue.py)
   3. **31. Batch Chai Preparation**
      1. 
      2. [my\_code\04\_loops\2\_batch\_chai.py](my_code/04_loops/2_batch_chai.py)
   4. **32. Looping through list - Orders Name**
      1. 
      2. [my\_code\04\_loops\3\_chai\_order\_with\_names.py](my_code/04_loops/3_chai_order_with_names.py)
   5. **33. Why to use Enumerate**
      1. 
      2. [my\_code\04\_loops\4\_tea\_menu.py](my_code/04_loops/4_tea_menu.py)
   6. **34. Zip Can Combine Lists**
      1. 
      2. [my\_code\04\_loops\5\_customer\_bill.py](my_code/04_loops/5_customer_bill.py)
   7. **35. Introducing While Loop in Python**
      1. 
      2. [my\_code\04\_loops\6\_simulate\_tea\_heating.py](my_code/04_loops/6_simulate_tea_heating.py)
   8. **36. Break, Continue and Loop Fallback**
      1. 
      2. [my\_code\04\_loops\07\_out\_of\_orders.py](my_code/04_loops/07_out_of_orders.py)
   9. **37. Walrus Operator(:=) is Interesting in Python**
      1. Introduced in Python 3.8.
      2. Called the walrus operator because := looks like walrus eyes and tusks.
      3. Allows assignment inside expressions → combines assignment and evaluation in one line.
      4. Eg:
         1. if (n := len("hello")) > 3:
         2. print(f"Length is {n}")
      5. Statements vs Expressions
         1. Statement: performs an action, does not return a value. Eg: x= 5
         2. Expression: returns a value when evaluated. Eg: 3 + 3
      6. Walrus operator lets you use assignment as part of an expression.
      7. Helps shorten code by combining assignment + evaluation.
      8. Important: Without walrus, attempting assignment inside expressions causes a syntax error.
      9. Code: [my\_code\04\_loops\09\_walrus\_operator.py](my_code/04_loops/09_walrus_operator.py)
   10. **38. Dictionary in place of Match Case**
       1. [my\_code\04\_loops\10\_dictionary\_case.py](my_code/04_loops/10_dictionary_case.py)
3. **Section 6: Functions in python**
   1. **39. Functions - Reducing Duplication and Splitting Complex Tasks**