

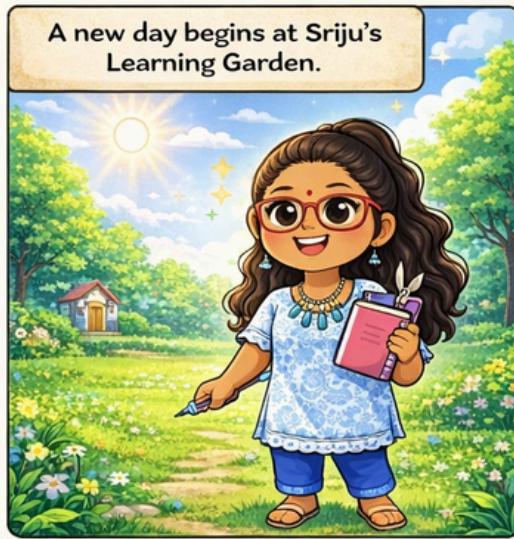
# INTERMEDIATE PYTHON ADVENTURES

VOLUME 2

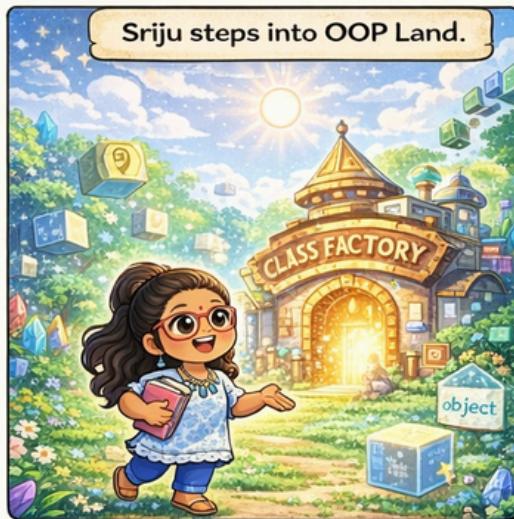


LEVEL UP YOUR PYTHON SKILLS!

A new day begins at Sriju's Learning Garden.



Sriju steps into OOP Land.



OOP means building things using objects.



Sriju: "So a class is like a blueprint?"



Everything around her forms from classes.



Sriju feels like she entered a coding universe.



Owl: "Classes create objects."



Sriju: "Let's build something!"



Sriju: "Challenge accepted!"



First step – define the class.

```
class PetBot:
```



PetBot starts to form...

```
class PetBot:
```

```
def __init__(self, name, color, size):  
    self.name = _____  
    self.color = blue  
    self.size = small
```

Sriju: "Let's give it a name!"

name = "Byte"

OK

Cancel

Owl: Now create the object!"

```
bot1 = PetBot("Byte")
```

Every class has an initializer.

```
def __init__(self, name, color, size):
```

- self.name = name
- self.color = color
- self.size = size

Give your object its features.

name = Byte

size = small



Attributes become part of the object.

name = Byte

size = small

Bot1 comes alive!

Beep boop!



Sriju's first custom object is ready.



Now that Bot1 exists, it needs actions.



Methods define what an object can do.

```
def speak(self):
```

Add more actions.

jump()  
dance()  
blink()

Bot1 becomes expressive.



Methods make objects useful.



Sriju: "Look! It can dance!"



Owl: "Self" the object itself.

```
let bot1 = new Robot()
bot1.speak()
```



Next, inheritance!

Create a SuperPetBot from PetBot.

PARENT

CHILD

class SuperPetBot (PetBot):

name = Byte size = small  
size = small size = blue

Create a SuperPetBot from PetBot.

Child class gets all parent features.

Add new powers!

Sensors

Laser Pointer

name = Byte  
size = small

name = Byte  
size = small

Now this is SUPER!

Override old behaviors.

Inheritance saves time.

PetBot:

Size = Super PetBot = ~~am~~ →

- ✓ Resuse Code
- ✓ Wall Bot
- ✓ SuperCed
- ✓ Sensors

I am ready!

Some features of an object must stay protected.



Private attributes keep important data safe.



Encapsulation prevents mistakes.



Now the robot works safely.



- `bot1.get_battery()`
- `bot1.set_battery(90)`

Owl: "This is called encapsulation."



Owl: "Use getters and setters to read or update safely."



Use getters and setters to read or update safely.



Encapsulation keeps objects healthy.



Protected      Safe.

Sriju opens her laptop... and something goes wrong.



Different errors appear as digital creatures.



Try-except catches errors before they cause trouble.



Each error type needs its own handler.



Else runs only when everything works fine.



Finally always runs at the end.



With handlers in place, workspace stabilizes.



Sriju discovers a studio where ideas multiply instantly.



A single brush stroke creates a whole list.



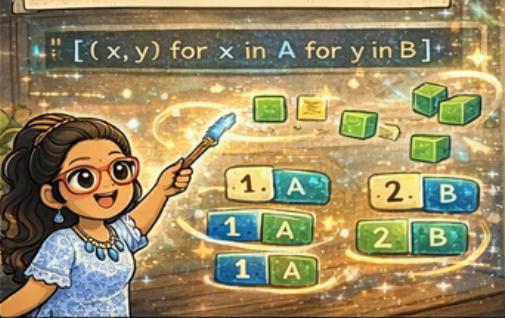
Comprehensions can modify items as they form.



Only keep the ones you want.



Even combinations are easy.



Build dictionaries in one smooth motion.



Less typing, more clarity.



With comprehensions, creating becomes effortless!



Sriju walks into a workshop where code powers machines.



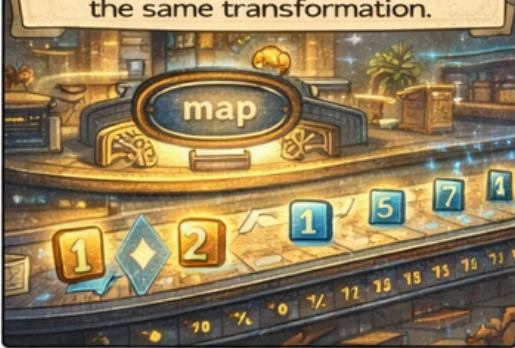
Tiny functions written in a single breath.



It transforms an item instantly



Map sends each item through the same transformation.



The whole row changes at once.



Filter lets only matching items through.



Together, they build powerful pipelines.



Sriju notices tall glowing info Towers outside her window.



APIs share information when you ask them correctly.



Python uses requests to talk to these towers.



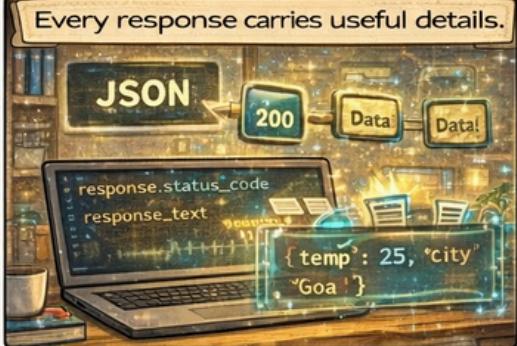
A GET request asks for information politely.



The tower sends back a stream of data



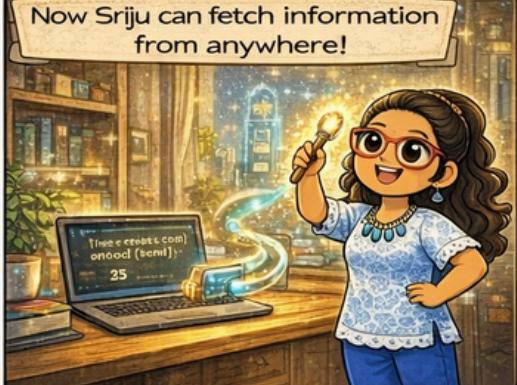
Every response carries useful details.



Sriju extracts only what she needs.



Now Sriju can fetch information from anywhere!



Sriju finds a room filled with glowing JSON crates.

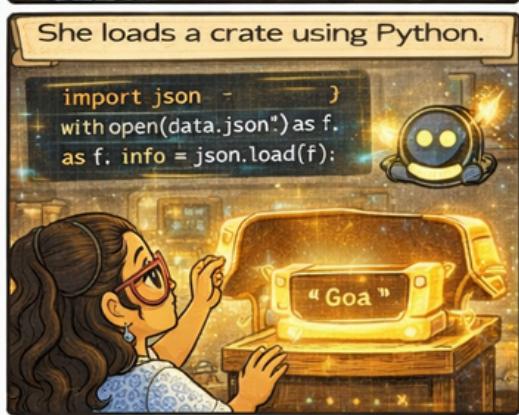


JSON stores information in a neat, readable format.

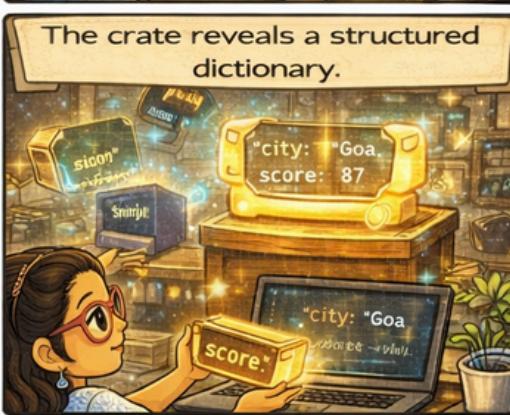


She loads a crate using Python.

```
import json  
with open('data.json') as f:  
    info = json.load(f)
```



The crate reveals a structured dictionary.



She picks out the exact piece she needs.



Sriju updates a value inside the crate.



She saves the updated data neatly..



Sriju now reads and writes JSON like a pro!



Sriju turns her workspace into a regex investigation room.



Regex helps find patterns hidden inside text.



Python uses the re module for regex work.



Then she collects every match.

`re.findall("cat", text)`

- . any character
- ^ starts with
- \$ ends with
- \d digits
- \w letters

Metacharacters have special powers.



Patterns help pick out numbers quickly.



Regex is powerful—use it carefully!



Sriju cracks the case with regex mastery!



Sriju enters a workspace where many tasks run at once.



## Multithreading

Threads share the same space but work in parallel.



Threads use the same memory to finish tasks faster.



Python can run threads with just a few lines.

```
import threading  
t1 = threading.Thread(target=task)  
t1.start()
```



Processes work separately with their own memory.

## Multiprocessing



Python's multiprocessing module starts new workers.

### Threads

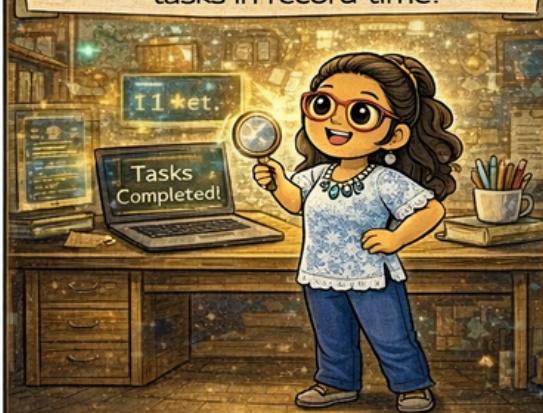
Shared memory

### Processes

Separate memories



With parallel helpers, Sriju completes tasks in record time!



A glowing Python Trial Orb appears in Sriju's workspace.



She builds a quick class to start the trial.



Next, she fetches and parses data flawlessly.



Her detective skills find the right pattern instantly.



Mini Sriju clones complete tasks in parallel.



The Python Trial Orb glows bright with approval



Sriju celebrates her achievement!



Volume 2 Complete!



**A Sriju Comic**



**Curiosity Lives Here!**

**Stories that smile back at you**