
INTERMEDIATE
PYTHON
ADVENTURES
VOLUME 2
Threads ✓
API ⚙
JSON ✓
REGEX
LEVEL UP YOUR PYTHON SKILLS!


A new day begins at Sriju's Learning Garden.
Suddenly... a glowing scroll!
It's from the Python Owl!
Exactly. Time for intermediate Python skills.
OOP
API
REGEX
This looks exciting!
Your next quest begins now!
OOP
API
Chapter One:
Enter – OOP Land!

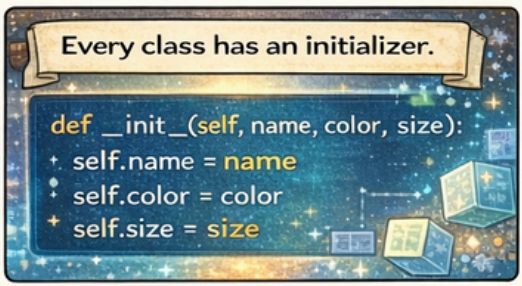

Sriju steps into OOP Land.
CLASS FACTORY
object
OOP means building things using objects.
color=blue
walk()
Sriju: "So a class is like a blueprint?"
RobotClass
Everything around her forms from classes.
size: = large
color = red
Sriju feels like she entered a coding universe.
Owl: "Classes create objects.
class PetBot:
__init__(self, name, color,
self.name = name
self.color = color
self.size = size
APPROVED
Sriju: "Let's build something!
Sriju: "Challenge accepted!"

First step — define the class.

```
class PetBot:
```

Every class has an initializer.

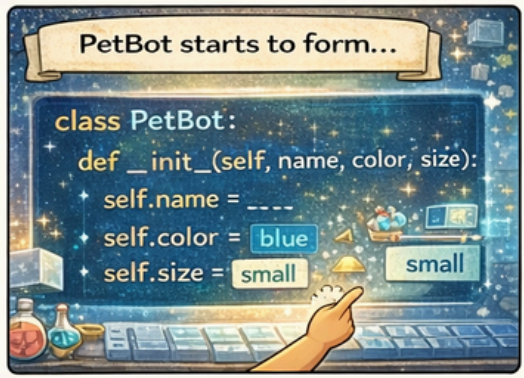
```
def __init__(self, name, color, size):
    self.name = name
    self.color = color
    self.size = size
```



PetBot starts to form...

```
class PetBot:
    def __init__(self, name, color, size):
        self.name = ____
        self.color = blue
        self.size = small
```

small



Give your object its features.

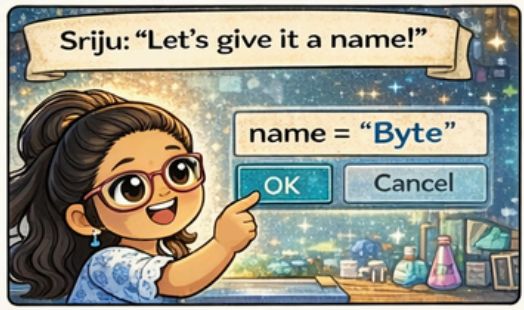
name = Byte

size = small

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

name = "Byte"

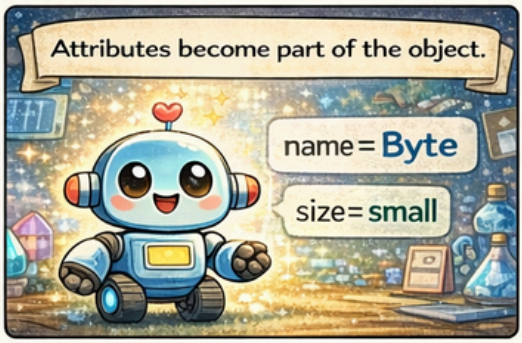
OK Cancel



Attributes become part of the object.

name = Byte

size = small



Owl: Now create the object!"

bot1 = PetBot("Byte")

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()


Let's make Bot1 speak!
Hello, World!


Bot1 becomes expressive.
Jump()
Let's dance!

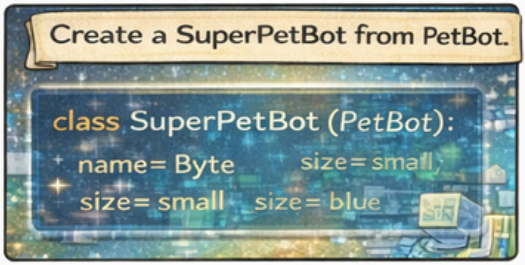

Methods make objects useful.
Doing!
Blink!

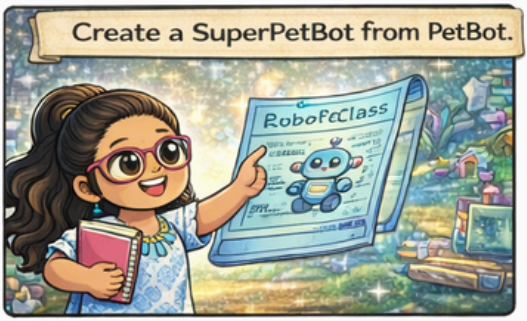

Sriju: "Look! It can dance!
Let's dance!
Boing!


Behaviors


Owl: "SelF" the object itself.
bot1 = Speak()


Next, inheritance!
PARENT
CHILD


Create a SuperPetBot from PetBot.
class SuperPetBot (PetBot):
name= Byte
size= small
size= small
size= blue


Create a SuperPetBot from PetBot.
RoboticClass


Child class gets all parent features.
name=Byte
size=small
name=Byte
size=small


Add new powers!
Laser Pointer
Sensors


Override old behaviors.


Now this is SUPER!


Inheritance saves time.
PetBot:
Size = Super PetBot
I am ready!


Some features of an object must stay protected.


Owl: “This is called encapsulation.
ENCAPSULATION
_private
__very_private


Private attributes keep important data safe.


Owl: “Use getters and setters to read or update safely.
bot1._battery_level = 200


Encapsulation prevents mistakes.
bot1.__battery_level = 200


Use getters and setters to read or update safely.
bot1.get_battery()
bot1.set_battery(90)
90%


Now the robot works safely.
bot1.get_battery()
bot1.set_battery(90)
Battery: 90%
90%


Encapsulation keeps objects healthy.
Protected
Safe.


Sriju opens her laptop... and something goes wrong.
ERROR!
Different errors appear as digital creatures.
TypeError!
ValueError
NameError!
hepeError
Try–except catches errors before they cause trouble.
ERROR TRACEBACK
File "main.py", line 3
<Error>
try:
run_task()
except
repair()
Each error type needs its own handler.
TypeError
ValueError
NameError
printt"All good"
Else runs only when everything works fine.
except ValueError
except ValueError
except hootfror
except pneserror
Finally always runs at the end.
finally: cleanup()
print"All good"
With handlers in place. workspace stabllizes.
NO ERRORS DETECTED


Sriju discovers a studio where ideas multiply instantly.
A single brush stroke creates a whole list.
[x*2 for x in items]
Comprehensions can modify items as they form.
[x*2 for x in numbers]
Only keep the ones you want.
[x for x in data if x:is.blue]
Even combinations are easy.
[(x, y) for x in A for y in B]
Build dictionaries in one smooth motion.
[ik:v for k, v in pairs]
10, 20
Star1
1. 2.
Star2
3. 3
Less typing, more clarity.
[x=2 for x in items]
Star1
Star1
Star2
Star3
With comprehensions, creating becomes effortless!
[x2 for x in items


Sriju walks into a workshop where code powers machines.
Tiny functions written in a single breath.
lambda x : x +1
lambda, x:
It transforms an item instantly
lambda,x : irto 4
Map sends each item through the same transformation.
map
The whole row changes at once.
Filter lets only matching items through.
filter
rotal
Together, they build powerful pipeiines.
lambda
map
relsut
map
corr


Sriju notices tall glowing info Towers outside her window.
APIs share information when you ask them correctly.
import requests
import requests
Python uses requests to talk to these towers.
import requests
GET
A GET request asks for information politely.
GIEY
The tower sends back a stream of data
GET
Every response carries useful details.
JSON
200
Data
Data!
response.status_code
response_text
{ temp : 25, city
Goa }
Sriju extracts only what she needs.
data = response.json()
print(data["temp"])
JSON
200
404
Not found
Now Sriju can fetch information from anywhere!
25


Sriju finds a room filled with glowing JSON crates.
JSON stores information in a neat, readable format.
"name": "Sriju"
"age": 21
She loads a crate using Python.
import json
with open("data.json") as f:
as f, info = json.load(f):
"Goa"
The crate reveals a structured dictionary.
"city": "Goa"
score: 87
She picks out the exact piece she needs
info["Goa"]
"Goa"
info["city"]
info[2]
Sriju updates a value inside the crate.
"score": 87
"score": 95
She saves the updated data neatly.
Sriju now reads and writes JSON like a pro!
JSON


Sriju turns her workspace into a regex investigation room.
Regex helps find patterns hidden inside text.
cat, bat, mat, hat
.*at
Python uses the re module for regex work.
cat, bat, mat, hat...
cat, bat, mat, ...
cat
bat
mat
FIND ALL EVIDENCE
Then she collects every match.
re.search("cat", text)
. any character
^ starts with
$ ends with
\d digits
\w letters
Metacharacters have special powers.
re.findall("...at", text)
cat
bat
mat
Patterns help pick out numbers quickly.
48291
Regex is powerful—use it carefully!
Regex com
any character
starts with
ends with
digits
letters
48291
48291
Sriju cracks the case with regex mastery!
CASE SOLVED


Page 13 — Multithreading & Multiprocessing (Intro)
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 rhemories
Threads
Shared Memory
Processes
Separatic Memories
With parallel helpers, Sriju completes tasks in record time!
Tasks Completed!


Page 15 — Final Boss & Completion Certificate
A glowing Python Trial Orb appears in Sriju's workspace.
Fix Error
Match Pattern
Fetch API
Use a Class
</>
She builds a quick class to start the trial.
class Demo:
Next, she fetches and parses data flawlessly.
{ data:... }
Her detective skills find the right pattern instantly.
Mini Sriju clones complete tasks in parallel.
Success
The Python Trial Orb glows bright with approval
Success
Sriju celebrates her achievement!
Intermediate Pythonist
You've earned the title...
Sriju!
Volume 2 Complete!