■ Python Lottery System

Coding Guide (SRP Edition)

Project Overview

You will build a **Lottery Simulation System** using Object-Oriented Programming in Python. This project emphasizes the **Single Responsibility Principle (SRP)** — each class and file should have only ONE job.

Lottery Rules:

- **Ticket Price:** $2 per ticket
- **Number Format:** 5 white balls (1-69) + 1 power ball (1-26)
- **Winning:** All 6 numbers must match to win the jackpot

Project Structure

```
lottery_project/
■
■■■ constants.py           # All constant values
■■■ validator.py           # Input validation logic
■■■ lottery_ticket.py      # Ticket data structure
■■■ lottery_machine.py     # Drawing & checking logic
■■■ player.py              # Player state management
■■■ display.py             # All output/display logic
■■■ main.py                # Program entry point
```

Each file has **ONE responsibility**. Let's break them down:

File 1: constants.py

Responsibility: Store all constant values in one place.

```
# constants.py

TICKET_PRICE = 2

WHITE_BALL_MIN = 1
WHITE_BALL_MAX = 69
WHITE_BALL_COUNT = 5

POWER_BALL_MIN = 1
POWER_BALL_MAX = 26
```

Why?

- If rules change (e.g., price becomes $3), you only edit ONE file
- No "magic numbers" scattered throughout your code

File 2: validator.py

Responsibility: Validate all user inputs.

Validator Class Methods:

Method	Description
__init__(self)	Initialize the validator
validate_white_balls(self, numbers)	Check if 5 unique numbers within 1-69. Return True/False
validate_power_ball(self, number)	Check if number is within 1-26
validate_ticket_count(self, count)	Check if count ≥ 1
validate_balance(self, balance, cost)	Check if player has enough money

Example:

```
from constants import *

class Validator:
    def __init__(self):
        pass

    def validate_white_balls(self, numbers):
        if len(numbers) != WHITE_BALL_COUNT:
            return False
        if len(set(numbers)) != WHITE_BALL_COUNT:
            return False  # duplicates exist
        for num in numbers:
            if not (WHITE_BALL_MIN <= num <= WHITE_BALL_MAX):
                return False
        return True

    def validate_power_ball(self, number):
        return POWER_BALL_MIN <= number <= POWER_BALL_MAX
```

File 3: lottery_ticket.py

Responsibility: Represent a single lottery ticket (data only).

Attributes:

- **white_balls** — List of 5 numbers (sorted)
- **power_ball** — Single number

LotteryTicket Class Methods:

Method	Description
__init__(self, white_balls, power_ball)	Initialize ticket
__str__(self)	Return formatted string (e.g., "05-12-23-45-67 | PB: 14")
get_white_balls(self)	Return white balls list
get_power_ball(self)	Return power ball

Important: This class should NOT:

- Generate random numbers (that's LotteryMachine's job)
- Validate numbers (that's Validator's job)
- Print anything directly (that's Display's job)

File 4: lottery_machine.py

Responsibility: Generate tickets and perform drawings.

Attributes:

- **winning_ticket** — The winning LotteryTicket (initially None)
- **validator** — A Validator object for input validation

LotteryMachine Class Methods:

Method	Description
__init__(self)	Initialize with no winning ticket, create a Validator
generate_random_ticket(self)	Create and return a random LotteryTicket
create_manual_ticket(self, white_balls, power_ball)	Create ticket from user input (use Validator first!)
draw_winning_numbers(self)	Generate and store winning ticket
get_winning_ticket(self)	Return the winning ticket
check_ticket(self, ticket)	Compare ticket to winning numbers, return match results

check_ticket Return Format:

```
{
    "white_matches": 3,      # how many white balls matched
    "power_match": True,     # did power ball match?
    "is_jackpot": False      # all 6 matched?
}
```

File 5: player.py

Responsibility: Manage player's money and ticket collection.

Attributes:

- **balance** — Current money
- **tickets** — List of purchased LotteryTicket objects

Player Class Methods:

Method	Description
__init__(self, initial_balance)	Set balance, empty ticket list

get_balance(self)	Return current balance
get_tickets(self)	Return list of tickets
get_ticket_count(self)	Return number of tickets
can_afford(self, amount)	Return True if balance ≥ amount
deduct_balance(self, amount)	Subtract from balance
add_ticket(self, ticket)	Add ticket to collection
buy_ticket(self, ticket)	Deduct $2 and add ticket (combine above two)

File 6: display.py

Responsibility: Handle ALL output to the user.

Display Class Methods:

Method	Description
__init__(self)	Initialize the display
show_welcome(self)	Print welcome message
show_ticket(self, ticket, number=None)	Print a single ticket
show_all_tickets(self, tickets)	Print all tickets with numbers
show_winning_numbers(self, ticket)	Print winning numbers with special formatting
show_result(self, ticket_num, result)	Print match result for one ticket
show_all_results(self, results)	Print all results
show_jackpot_winner(self)	Print jackpot celebration message
show_no_winner(self)	Print "no winner" message
show_summary(self, ticket_count, spent, remaining)	Print final summary
show_error(self, message)	Print error message

Why separate Display?

- Easy to change output format later
- Could swap to GUI without changing other classes
- All user-facing text in one place

File 7: main.py

Responsibility: Coordinate all classes and run the program.

Structure:

```
from constants import TICKET_PRICE
from validator import Validator
```

```
from lottery_ticket import LotteryTicket
from lottery_machine import LotteryMachine
from player import Player
from display import Display

def get_ticket_choice():
    """Ask user for manual or random ticket"""
    # Get input, return 'M' or 'R'
    pass

def get_manual_numbers():
    """Get and validate manual number input from user"""
    # Use Validator to check input
    # Return (white_balls, power_ball) tuple
    pass

def main():
    # Create helper objects
    display = Display()
    machine = LotteryMachine()

    # 1. Show welcome
    display.show_welcome()

    # 2. Get starting balance from user
    # 3. Create Player

    # 4. Purchase phase
    #    - Ask how many tickets
    #    - Loop: get each ticket (manual/random)
    #    - Add to player

    # 5. Show all purchased tickets
    display.show_all_tickets(player.get_tickets())

    # 6. Draw winning numbers
    machine.draw_winning_numbers()
    display.show_winning_numbers(machine.get_winning_ticket())

    # 7. Check all tickets, collect results

    # 8. Display results

    # 9. Show summary

if __name__ == "__main__":
    main()
```

Class Interaction Diagram

```
main.py (coordinator)
    ■
    ■■■ creates ■■→ Player
    ■                ■■■ stores ■■→ LotteryTicket (multiple)
    ■
    ■■■ creates ■■→ LotteryMachine
    ■                ■■■ has ■■■■→ Validator
    ■                ■■■ generates ■■→ LotteryTicket
    ■                ■■■ stores ■■→ LotteryTicket (winning)
    ■
    ■■■ creates ■■→ Display
    ■
    ■■■ imports ■→ constants
```

Import Guide

File	Imports

constants.py	(none)
validator.py	constants
lottery_ticket.py	(none) or constants for formatting
lottery_machine.py	constants, validator, lottery_ticket, random
player.py	constants
display.py	(none)
main.py	ALL of the above

Example Output

```
========================================
       WELCOME TO LOTTERY SIMULATOR
========================================

Enter your starting balance: $20

How many tickets would you like to buy? 2

--- Ticket 1 ---
Manual (M) or Random (R)? R
Generated: 08-15-23-45-62 | PB: 19

--- Ticket 2 ---
Manual (M) or Random (R)? M
Enter 5 white ball numbers (1-69): 5 10 15 20 25
Enter power ball number (1-26): 7
Your ticket: 05-10-15-20-25 | PB: 07

========================================
            YOUR TICKETS
========================================
1. 08-15-23-45-62 | PB: 19
2. 05-10-15-20-25 | PB: 07

========================================
      DRAWING WINNING NUMBERS...
========================================
Winning Numbers: 05-10-22-33-68 | PB: 07

========================================
              RESULTS
========================================
Ticket 1: 0 white balls | Power Ball: NO
Ticket 2: 2 white balls | Power Ball: YES

No jackpot winner this time!

========================================
              SUMMARY
========================================
Tickets purchased: 2
Money spent: $4
Remaining balance: $16
```

Common Mistakes to Avoid

❌ Wrong	✅ Right
Printing inside LotteryTicket	Use Display class for all output

Hardcoding 69 or 2 in code	Import from constants.py
Validating inside LotteryTicket	Use Validator class
Generating randoms inside Player	Use LotteryMachine

Useful Python Tips

Concept	Example
f-string padding	f"{num:02d}" → "05"
Random unique	random.sample(range(1,70), 5)
Set intersection	set(a) & set(b)
Dictionary return	return {"key": value}
Create object	display = Display()

Good luck! ■