**Solution 1: Squirtle Class Implementation**

class Squirtle:

"""

A class representing a Squirtle Pokémon with health and power management.

Attributes:

hp (int): Health points, initialized to 44

pp (int): Power points, initialized to 300

"""

def \_\_init\_\_(self):

"""Initialize Squirtle with default HP and PP values."""

self.hp = 44

self.pp = 300

def water\_gun(self):

"""Use Water Gun attack, consuming 25 PP."""

if self.pp >= 25:

self.pp -= 25

print("Used Water Gun! -25 PP")

else:

print("Not enough PP for Water Gun!")

def bite(self):

"""Use Bite attack, consuming 25 PP."""

if self.pp >= 25:

self.pp -= 25

print("Used Bite! -25 PP")

else:

print("Not enough PP for Bite!")

def skull\_bash(self):

"""Use Skull Bash attack, consuming 10 PP."""

if self.pp >= 10:

self.pp -= 10

print("Used Skull Bash! -10 PP")

else:

print("Not enough PP for Skull Bash!")

def hydro\_pump(self):

"""Use Hydro Pump attack, consuming 5 PP."""

if self.pp >= 5:

self.pp -= 5

print("Used Hydro Pump! -5 PP")

else:

print("Not enough PP for Hydro Pump!")

def being\_attacked(self, damage):

"""

Take damage from an attack.

Args:

damage (int): Amount of HP to subtract

"""

self.hp -= damage

print(f"Ouch! Took {damage} damage!")

if self.hp <= 0:

self.hp = 0

print("Squirtle fainted! Needs treatment!")

def treated\_at\_pokemon\_center(self):

"""Fully restore HP and PP at Pokémon Center."""

self.hp = 44

self.pp = 300

print("Fully healed at Pokémon Center!")

def needs\_treatment(self):

"""Check if Squirtle needs treatment (HP = 0)."""

return self.hp == 0

**Solution 2: CSV File I/O Exercise**

import csv

class Student:

"""

A class representing a student with their academic records.

Attributes:

student\_id (str): Student identification number

name (str): Student name

grades (dict): Dictionary of subject:grade pairs

"""

def \_\_init\_\_(self, student\_id, name, grades=None):

"""

Initialize a Student instance.

Args:

student\_id (str): Student ID

name (str): Student name

grades (dict): Dictionary of grades (default: empty dict)

"""

self.student\_id = student\_id

self.name = name

self.grades = grades if grades else {}

def add\_grade(self, subject, grade):

"""

Add a grade for a specific subject.

Args:

subject (str): Subject name

grade (str): Grade received

"""

self.grades[subject] = grade

def calculate\_average(self):

"""

Calculate the average grade across all subjects.

Returns:

float: Average grade (or 0 if no grades)

"""

if not self.grades:

return 0.0

numeric\_grades = []

for grade in self.grades.values():

try:

numeric\_grades.append(float(grade))

except ValueError:

continue

return sum(numeric\_grades) / len(numeric\_grades) if numeric\_grades else 0.0

class ComputerScienceClass:

"""

A class representing a computer science class with multiple students.

Attributes:

students (list): List of Student objects

"""

def \_\_init\_\_(self):

"""Initialize an empty class."""

self.students = []

def add\_student(self, student):

"""

Add a student to the class.

Args:

student (Student): Student object to add

"""

self.students.append(student)

def sort\_students\_by\_id(self):

"""Sort students by their ID in ascending order."""

self.students.sort(key=lambda x: x.student\_id)

def save\_reformatted\_csv(self, output\_filename):

"""

Save student records to a reformatted CSV file.

Args:

output\_filename (str): Path to output CSV file

"""

if not self.students:

return

self.sort\_students\_by\_id()

# Get all unique subjects across all students

all\_subjects = set()

for student in self.students:

all\_subjects.update(student.grades.keys())

all\_subjects = sorted(all\_subjects)

with open(output\_filename, 'w', newline='') as csvfile:

writer = csv.writer(csvfile)

# Write header row (Student ID, Name, Subjects)

writer.writerow(['Student ID', 'Name'] + all\_subjects + ['Average'])

# Write each student's data

for student in self.students:

row = [student.student\_id, student.name]

for subject in all\_subjects:

row.append(student.grades.get(subject, ''))

row.append(f"{student.calculate\_average():.2f}")

writer.writerow(row)

def process\_student\_records(input\_file, output\_file):

"""

Process student records from input CSV to reformatted output CSV.

Args:

input\_file (str): Path to input CSV file

output\_file (str): Path to output CSV file

"""

cs\_class = ComputerScienceClass()

with open(input\_file, 'r') as csvfile:

reader = csv.DictReader(csvfile)

for row in reader:

student\_id = row.get('Student ID', '').strip()

name = row.get('Name', '').strip()

if not student\_id or not name:

continue

student = Student(student\_id, name)

# Add all grades from the row (excluding ID and Name)

for key, value in row.items():

if key not in ['Student ID', 'Name'] and value.strip():

student.add\_grade(key.strip(), value.strip())

cs\_class.add\_student(student)

cs\_class.save\_reformatted\_csv(output\_file)

# Example usage

process\_student\_records('student\_record.csv', 'student\_record\_reformatted.csv')