# === Fully Defined Agent Functions ===

def creative\_collaboration\_agent():

return "Creative Agent: Co-creating art, music, and literature with human partners."

def simulation\_agent():

return "Simulation Agent: Running 'what-if' scenarios and predictive modeling."

def ethics\_agent():

return "Ethics Agent: Ensuring all actions align with moral and societal guidelines."

def memory\_agent():

return "Memory Agent: Storing, retrieving, and learning from past interactions."

def research\_agent():

return "Research Agent: Gathering and synthesizing up-to-date information."

def default\_agent():

return "Default Agent: General purpose response."

# === Genome-Inspired Agent Mapping ===

genome\_agents = {

'chr1': creative\_collaboration\_agent,

'chr2': simulation\_agent,

'chr3': ethics\_agent,

'chr4': memory\_agent,

'chr5': research\_agent,

# Assign default\_agent to remaining chromosomes for simplicity, can be extended

'chr6': default\_agent, 'chr7': default\_agent, 'chr8': default\_agent, 'chr9': default\_agent, 'chr10': default\_agent,

'chr11': default\_agent, 'chr12': default\_agent, 'chr13': default\_agent, 'chr14': default\_agent, 'chr15': default\_agent,

'chr16': default\_agent, 'chr17': default\_agent, 'chr18': default\_agent, 'chr19': default\_agent, 'chr20': default\_agent,

'chr21': default\_agent, 'chr22': default\_agent, 'chrX': default\_agent, 'chrY': default\_agent

}

# === AdvancedCodetteAI Class Using Genome Map ===

class GenomicCodetteAI:

def \_\_init\_\_(self, genome\_map, max\_retries=2):

self.genome\_map = genome\_map

self.max\_retries = max\_retries

def generate\_response(self, chromosome, attempt=0):

agent = self.genome\_map.get(chromosome, self.genome\_map.get('chr1')) # Fallback to chr1 if not found

response = agent()

if response is None:

if attempt < self.max\_retries:

# Try next chromosome

next\_chr = self.\_next\_fallback\_chromosome(chromosome)

return self.generate\_response(next\_chr, attempt + 1)

else:

return "System: Exhausted all self-healing attempts. Please try a new prompt or check system inputs."

return response

def \_next\_fallback\_chromosome(self, current\_chr):

# Cycle through available chromosomes for fallback

fallback\_chrs = list(self.genome\_map.keys())

if current\_chr in fallback\_chrs:

idx = fallback\_chrs.index(current\_chr)

next\_idx = (idx + 1) % len(fallback\_chrs)

return fallback\_chrs[next\_idx]

else:

return 'chr1'

def visualize\_genome(self):

# Show which agent is mapped to which chromosome

lines = ["Codette AI Genome Map:"]

for chr\_name, agent in self.genome\_map.items():

agent\_name = agent.\_\_name\_\_

lines.append(f"{chr\_name}: {agent\_name}")

return "\n".join(lines)

# === Instantiate and Demonstrate ===

genomic\_codette\_ai = GenomicCodetteAI(genome\_agents)

# Visualize the AI "genome"

print(genomic\_codette\_ai.visualize\_genome())

# Generate responses from different "chromosomal" agents

print(genomic\_codette\_ai.generate\_response('chr1')) # Creative Agent

print(genomic\_codette\_ai.generate\_response('chr2')) # Simulation Agent

print(genomic\_codette\_ai.generate\_response('chr3')) # Ethics Agent

print(genomic\_codette\_ai.generate\_response('chr4')) # Memory Agent

print(genomic\_codette\_ai.generate\_response('chr5')) # Research Agent

print(genomic\_codette\_ai.generate\_response('chrX')) # Default Agent

# This code is fully functional and can be extended with more specialized agents or richer genome logic as desired.