Class Notes

ANNOUNCEMENTS CAN BE FOUND AT:

**Review of Rabbits and Wolves Week 1-2 turn in.*:

- Always include a table
- Can resubmit
- INCLUDE THE PSUEDO DESIGN IN R&W Part 2

Expectation for coding R&W Part 2:

1. Problem statement
2. Summary statement of logic (input processing output paragraph)
3. Detailed pseudo-code
4. processing of logic on test data based on results (must match expected outcomes with actual outcomes)
5. Test everything
6. Instructions?
7. Video

GOOD PRACTICES:

- init variables to 0 (must start somewhere also helps logically)
- simplify logic to variables do not over complicate it
- planning is perfection
- test every loop / decision and the points to decisions

Break @ 1210

Continue @ 1220

```
// FUNCTION simulate_population
// This function simulates the population dynamics of rabbits and wolves
// on an island over a 20-year period, following the specific set of rules
// for population growth, predation, and death rates as defined by the
project.

// Constants
SET INITIAL_RABBITS to 50
SET RABBIT_GROWTH_RATE to 0.10
SET WOLF_GROWTH_RATE to 0.08
```

```
SET WOLF_DEATH_RATE to 0.06
SET PREDATION_RATE to 0.01
SET WOLF_INTRODUCTION_YEAR to 5
SET INITIAL_WOLVES_COUNT to 10
SET SIMULATION_YEARS to 20

// Initialization
SET rabbits to INITIAL_RABBITS
SET wolves to 0

// Initial Output
PRINT " Year | Rabbits | Wolves "
PRINT "-------------------------"
PRINT " 0 | 50 | 0"

// Simulation Loop
FOR year FROM 1 TO SIMULATION_YEARS
    // 1. Rabbit population growth
    COMPUTE rabbits as rabbits * (1 + RABBIT_GROWTH_RATE)

    // 2. Wolf introduction
    IF year IS EQUAL TO WOLF_INTRODUCTION_YEAR THEN
        SET wolves to INITIAL_WOLVES_COUNT
    ENDIF

    // 3. Rabbit loss due to predation
    IF wolves > 0 THEN
        COMPUTE rabbit_loss as rabbits * PREDATION_RATE * wolves
        COMPUTE rabbits as rabbits - rabbit_loss
    ENDIF

    // 4. Wolf population change
    IF wolves > 0 THEN
        COMPUTE net_wolf_growth_rate as WOLF_GROWTH_RATE - WOLF_DEATH_RATE
        COMPUTE wolves as wolves * (1 + net_wolf_growth_rate)
    ENDIF

    // 5. Enforce integer populations
    COMPUTE rabbits as INTEGER(rabbits)
    COMPUTE wolves as INTEGER(wolves)

    // 6. Prevent negative populations
    IF rabbits < 0 THEN SET rabbits to 0
    IF wolves < 0 THEN SET wolves to 0

    // 7. Display current year results
```

```
    PRINT year, rabbits, wolves
ENDFOR

// END FUNCTION
```

```python
# Owen Lindsey
# Rabbits and Wolves
# Due: 09/28/2025

def simulate_population(
    initial_rabbits=50,         # Rabbits at year 0
    initial_wolves=0,           # Wolves at year 0 (introduced later)
    rabbit_growth_rate=0.10,    # 10% rabbit growth
    wolf_growth_rate=0.08,      # 8% wolf growth
    wolf_death_rate=0.06,       # 6% wolf death
    predation_rate=0.01,        # 1% predation rate per wolf
    wolf_introduction_year=5,   # Year wolves are introduced
    initial_wolf_count=10,      # Number of wolves introduced
    simulation_years=20         # Total years to simulate
):
    """
    Simulates rabbit and wolf population dynamics.
    Returns a list of (year, rabbits, wolves).
    Raises ValueError if invalid parameters are provided.
    """

    # Validate inputs
    if initial_rabbits < 0 or initial_wolves < 0:
        raise ValueError("Starting populations must be non-negative.")
    if rabbit_growth_rate < 0 or wolf_growth_rate < 0 or wolf_death_rate < 0
or predation_rate < 0:
        raise ValueError("Growth/death rates must be non-negative.")
    if simulation_years <= 0:
        raise ValueError("Simulation years must be positive.")

    # Initialize state
    rabbits = initial_rabbits
    wolves = initial_wolves
    results = []

    # Print header
    print(f'{"Year":^6}{"Rabbits":^12}{"Wolves":^12}')
    print('-' * 30)
```

```
    # Year 0
    print(f'{0:^6}{rabbits:^12}{wolves:^12}')
    results.append((0, rabbits, wolves))

    # Simulation loop
    for year in range(1, simulation_years + 1):
        # Rabbits grow
        rabbits = int(rabbits * (1 + rabbit_growth_rate))

        # Wolves introduced
        if year == wolf_introduction_year:
            wolves = initial_wolf_count

        # Predation + wolf growth
        if wolves > 0:
            rabbit_loss = int(rabbits * predation_rate * wolves)
            rabbits -= rabbit_loss

            net_wolf_growth = wolf_growth_rate - wolf_death_rate
            wolves = int(wolves * (1 + net_wolf_growth))

        # Prevent negative values
        rabbits = max(rabbits, 0)
        wolves = max(wolves, 0)

        # Print and store results
        print(f'{year:^6}{rabbits:^12}{wolves:^12}')
        results.append((year, rabbits, wolves))

    return results


# --- Testing with Expected Results ---
try:
    # Default run (rabbits grow until year 5, then wolves arrive)
    print("\nDefault Simulation Run:\n")
    default_results = simulate_population()

    # EXPECTED (first few years, rounded to ints):
    # Year 0: Rabbits=50, Wolves=0
    # Year 1: Rabbits=55, Wolves=0
    # Year 2: Rabbits=60, Wolves=0
    # Year 3: Rabbits=66, Wolves=0
    # Year 4: Rabbits=72, Wolves=0
    # Year 5: Rabbits drops after wolves introduced (approx 79 → 71),
Wolves=10
```

```
    # Year 6: Rabbits ~70, Wolves grow slightly (~10.2 → 10)
    # By Year 20, wolves stabilize around teens, rabbits keep shrinking under
predation.

    # Modified run with more wolves earlier
    print("\nModified Simulation Run (100 rabbits, wolves at year 2, 5
wolves):\n")
    mod_results = simulate_population(initial_rabbits=100,
wolf_introduction_year=2, initial_wolf_count=5)

    # EXPECTED:
    # Year 0: Rabbits=100, Wolves=0
    # Year 1: Rabbits=110, Wolves=0
    # Year 2: Wolves introduced, Rabbits drop (121 → ~115), Wolves=5
    # Year 3: Rabbits continue but under pressure, Wolves grow (~5.1 → 5)
    # By Year 10, rabbits much lower than default run.

    # Invalid run (should fail)
    print("\nInvalid Test Run (expect error):\n")
    simulate_population(initial_rabbits=-10)  # invalid input

except ValueError as e:
    # Catch intended validation errors
    print(f"Simulation failed: {e}")

except Exception as e:
    # Catch unexpected errors
    print(f"Unexpected error: {e}")
```

Year Rabbits Wolves

0 50 0
1 55 0
2 60 0
3 66 0
4 72 0
5 72 10
6 72 10
7 72 10
8 72 10
9 72 10
10 72 10
11 72 10

12 72 10
13 72 10
14 72 10
15 72 10
16 72 10
17 72 10
18 72 10
19 72 10
20 72 10

Modified Simulation Run (100 rabbits, wolves at year 2, 5 wolves):

Year Rabbits Wolves

0 100 0
1 110 0
2 115 5
3 120 5
4 126 5
5 132 5
6 138 5
7 144 5
8 151 5
9 158 5
10 165 5
11 172 5
12 180 5
13 189 5
14 197 5
15 206 5
16 215 5
17 225 5
18 235 5
19 246 5
20 257 5

Invalid Test Run (expect error):

Simulation failed: Starting populations must be non-negative.

```
**War card game:**

 Remember the above ^^^
```