**Programming process steps in fix\_1\_haplo\_min.py**

**Steps carried out chronologically**

1. Import necessary Python modules such as numpy, os, sys, time, multiprocessing, and yaml.

2. Load configuration settings from the file "config\_1\_haplo\_min\_v1.yaml".

3. Extract the values for "Repetitions" and "max\_generations" from the configuration.

4. Define constants including MASTER\_SEED, input/output filenames, and example parameter rows.

5. Prevent system sleep on Windows using Windows API calls.

6. Record the program start time.

7. Define a function returning the expected input file header string.

8. Define a function returning the detailed results file header string.

9. If the input file does not exist, create it with headers and example rows, then exit.

10. Read all non-empty lines from the input file.

11. If the input file is empty, repopulate it with headers and example rows, then exit.

12. If the first line does not match the expected header, replace the file and exit.

13. If the input file contains only the header, append example rows and exit.

14. Validate that each data line has exactly four semicolon-separated fields.

15. Validate that population size (N) is a positive integer within the allowed range.

16. Validate that the selection coefficient (s) is a float between –2 and 2.

17. Validate that the initial allele frequency (p0) is a float between 0.0 and 1.0.

18. Validate that the number of attempts is a positive integer within the allowed range.

19. Store valid parameter sets in a list; exit if any errors are found.

20. Define the simulate\_population function to model allele dynamics under selection and drift.

21. Initialize counters for allele loss, fixation, and a list for fixation generations.

22. For each attempt, simulate up to max\_generations, updating allele frequency each generation.

23. Compute genotype frequencies assuming Hardy–Weinberg equilibrium.

24. Assign fitness values based on the selection coefficient s.

25. Calculate mean fitness and post-selection allele frequency.

26. Apply genetic drift via binomial sampling of allele counts.

27. Record loss if allele frequency hits 0; record fixation and generation if near fixation.

28. Define a worker function that sets a unique random seed and runs simulate\_population.

29. Determine the number of available CPU cores.

30. Generate a job list for all parameter-repetition combinations.

31. Execute jobs in parallel using multiprocessing and collect results.

32. Sort results by simulation number and repetition.

33. Aggregate raw results across repetitions for each parameter set using a defaultdict.

34. Sum losses, fixations, and fixation generations; store parameter metadata.

35. Compute pooled statistics: fixation/loss probabilities, standard error, and mean/std of fixation time.

36. Format individual-repetition result lines with per-rep statistics.

37. Delete any existing individual results file and write a new one with headers and data.

38. Skip writing the individual results file if access is denied and print a warning.

39. If Repetitions > 1, write pooled average results to "out\_avg\_1\_haplo\_min.txt".

40. Print total execution time.