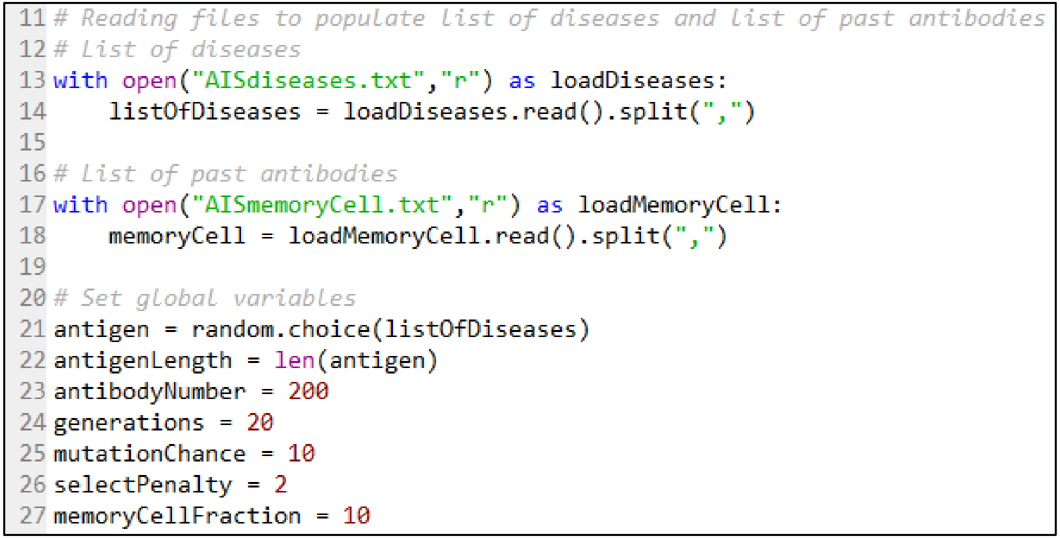
Systems Biology – Exercises

Week 13: Artificial Immune System

Artificial Immune System (AIS)

- As we have seen in the logic behind an artificial immune system, principles of evolution—mutation, fitness (or penalty), selection—are combined with a memory cell and vaccination capabilities.
- We will implement the notion of a clonal selection algorithm of an AIS. Refer to the working sheet from the lecture for technical terms. The code below sets the framework to load diseases and antibodies of previous immune responses. Also import random, string, numpy as np.

Code 1. Initialization of Diseases and Past Antibodies.

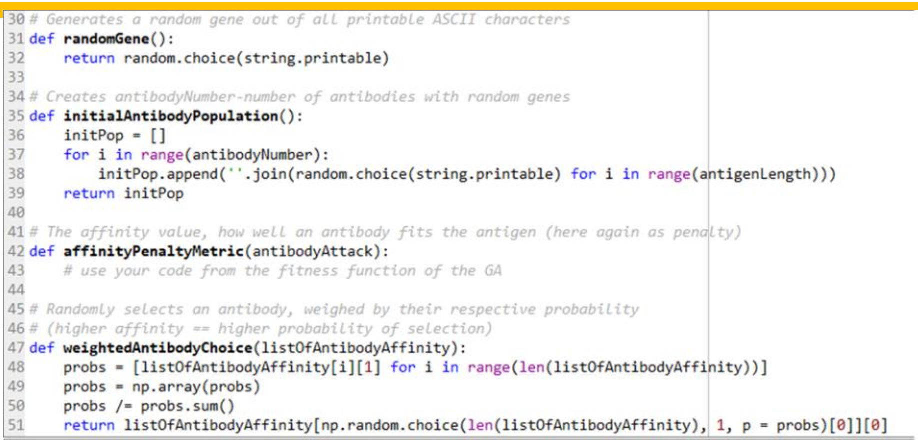


```python
# Reading files to populate list of diseases and list of past antibodies
# List of diseases
with open("AISdiseases.txt","r") as loadDiseases:
    listOfDiseases = loadDiseases.read().split(",")

# List of past antibodies
with open("AISmemoryCell.txt","r") as loadMemoryCell:
    memoryCell = loadMemoryCell.read().split(",")

# Set global variables
antigen = random.choice(listOfDiseases)
antigenLength = len(antigen)
antibodyNumber = 200
generations = 20
mutationChance = 10
selectPenalty = 2
memoryCellFraction = 10
```

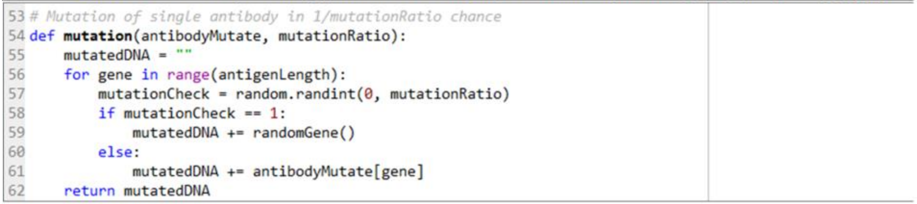
Mechanisms in AIS

- We see similarities with the genetic algorithm from previous classes and will modify the code to work as an artificial immune system.

Code 2. Functions and Mechanisms of the AIS



```python
# Generates a random gene out of all printable ASCII characters
def randomGene():
    return random.choice(string.printable)

# Creates antibodyNumber-number of antibodies with random genes
def initialAntibodyPopulation():
    initPop = []
    for i in range(antibodyNumber):
        initPop.append(''.join(random.choice(string.printable) for i in range(antigenLength)))
    return initPop

# The affinity value, how well an antibody fits the antigen (here again as penalty)
def affinityPenaltyMetric(antibodyAttack):
    # use your code from the fitness function of the GA

# Randomly selects an antibody, weighed by their respective probability
# (higher affinity == higher probability of selection)
def weightedAntibodyChoice(listOfAntibodyAffinity):
    probs = [listOfAntibodyAffinity[i][1] for i in range(len(listOfAntibodyAffinity))]
    probs = np.array(probs)
    probs /= probs.sum()
    return listOfAntibodyAffinity[np.random.choice(len(listOfAntibodyAffinity), 1, p = probs)[0]][0]
```

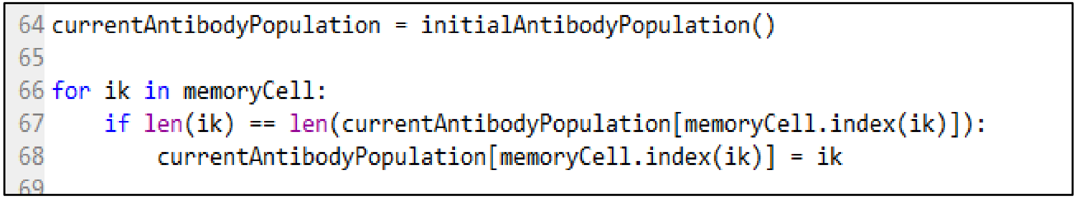


```python
# Mutation of single antibody in 1/mutationRatio chance
def mutation(antibodyMutate, mutationRatio):
    mutatedDNA = ""
    for gene in range(antigenLength):
        mutationCheck = random.randint(0, mutationRatio)
        if mutationCheck == 1:
            mutatedDNA += randomGene()
        else:
            mutatedDNA += antibodyMutate[gene]
    return mutatedDNA
```

Exercise 1

- From line “**def affinityPenaltyMetric(antibodyAttack):**” in the above algorithm, implement your version of the fitness function from the genetic algorithm.

Code 3. Reapplying Past Antibodies

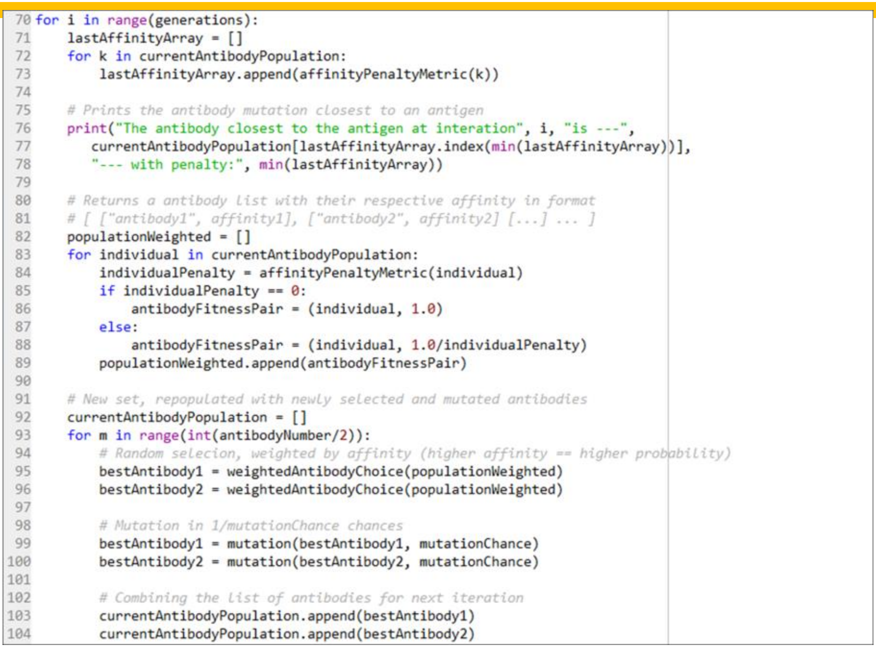


```python
currentAntibodyPopulation = initialAntibodyPopulation()

for ik in memoryCell:
    if len(ik) == len(currentAntibodyPopulation[memoryCell.index(ik)]):
        currentAntibodyPopulation[memoryCell.index(ik)] = ik
```

Exercise 2

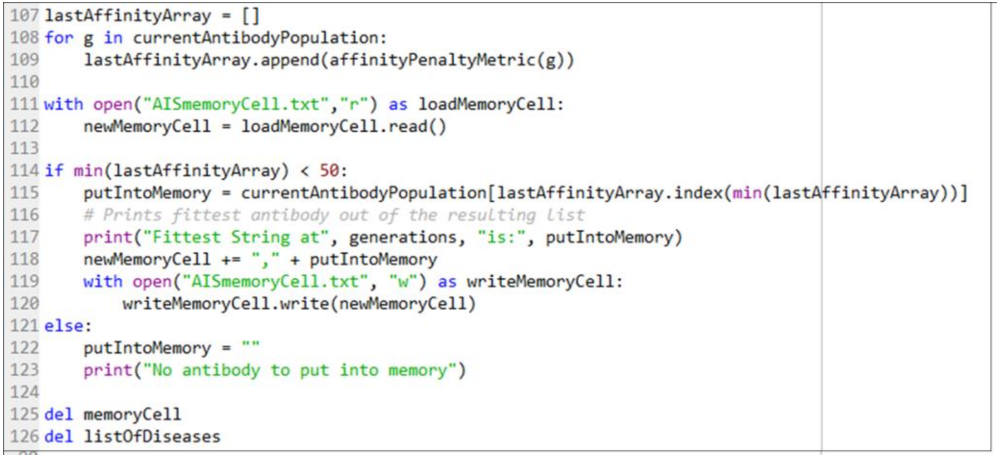
- In the code on page 7, the loop only works as long as the elements in the memory cell are less than the population of antibodies. Devise a way to limit the memory size to a fraction of the antibody population. Use memoryCellFraction so that the memory cell size is memoryCellFraction-times smaller. As new elements are added to the memory cell, old antibodies should be removed. This is called *aging* and limits the lifespan of antibodies.

Code 4. AIS Implementation



```python
for i in range(generations):
    lastAffinityArray = []
    for k in currentAntibodyPopulation:
        lastAffinityArray.append(affinityPenaltyMetric(k))

    # Prints the antibody mutation closest to an antigen
    print("The antibody closest to the antigen at interation", i, "is ---",
       currentAntibodyPopulation[lastAffinityArray.index(min(lastAffinityArray))],
       "--- with penalty:", min(lastAffinityArray))

    # Returns a antibody list with their respective affinity in format
    # [ ["antibody1", affinity1], ["antibody2", affinity2] [...] ... ]
    populationWeighted = []
    for individual in currentAntibodyPopulation:
        individualPenalty = affinityPenaltyMetric(individual)
        if individualPenalty == 0:
            antibodyFitnessPair = (individual, 1.0)
        else:
            antibodyFitnessPair = (individual, 1.0/individualPenalty)
        populationWeighted.append(antibodyFitnessPair)

    # New set, repopulated with newly selected and mutated antibodies
    currentAntibodyPopulation = []
    for m in range(int(antibodyNumber/2)):
        # Random selecion, weighted by affinity (higher affinity == higher probability)
        bestAntibody1 = weightedAntibodyChoice(populationWeighted)
        bestAntibody2 = weightedAntibodyChoice(populationWeighted)

        # Mutation in 1/mutationChance chances
        bestAntibody1 = mutation(bestAntibody1, mutationChance)
        bestAntibody2 = mutation(bestAntibody2, mutationChance)

        # Combining the list of antibodies for next iteration
        currentAntibodyPopulation.append(bestAntibody1)
        currentAntibodyPopulation.append(bestAntibody2)
```

Code 5. Testing Antibody Affinity and Addition to Memory



```python
lastAffinityArray = []
for g in currentAntibodyPopulation:
    lastAffinityArray.append(affinityPenaltyMetric(g))

with open("AISmemoryCell.txt","r") as loadMemoryCell:
    newMemoryCell = loadMemoryCell.read()

if min(lastAffinityArray) < 50:
    putIntoMemory = currentAntibodyPopulation[lastAffinityArray.index(min(lastAffinityArray))]
    # Prints fittest antibody out of the resulting list
    print("Fittest String at", generations, "is:", putIntoMemory)
    newMemoryCell += "," + putIntoMemory
    with open("AISmemoryCell.txt", "w") as writeMemoryCell:
        writeMemoryCell.write(newMemoryCell)
else:
    putIntoMemory = ""
    print("No antibody to put into memory")

del memoryCell
del listOfDiseases
```

Exercise 3

- Create two text files AISdiseases.txt and AISmemoryCell.txt in the same directory as your Python file. AISmemoryCell.txt should be empty at the beginning and will be populated with successful immune responses (antibodies). AISdiseases.txt must be in the form disease1,disease2,... and must be written verbatim in plain text. Look online for lists of viruses, bacteria, or parasites and put them in your file. Aim for at least 100 different diseases. More technical names make for a better representation of a disease’s genetic structure, but any name will do.

Exercise 4

- Create a new text file named AISvaccines.txt. This file can be populated with a list of disease names, representing a cocktail of vaccines. Modify your code so that the files AISvaccines.txt can be loaded, also the names in the vaccine will get attached to the beginning of the memory cell (with possible deletion of old antibodies in that list). Test your implementation of a vaccination campaign by using a list of 10 diseases in **AISdiseases_4.txt** with similar but different names and adding 5 out of these 10 into the vaccination cocktail (AISvaccines.txt).

- **No Homework in week 13**

Systems Biology – Exercises

Week 14:

Epidemiological Modeling