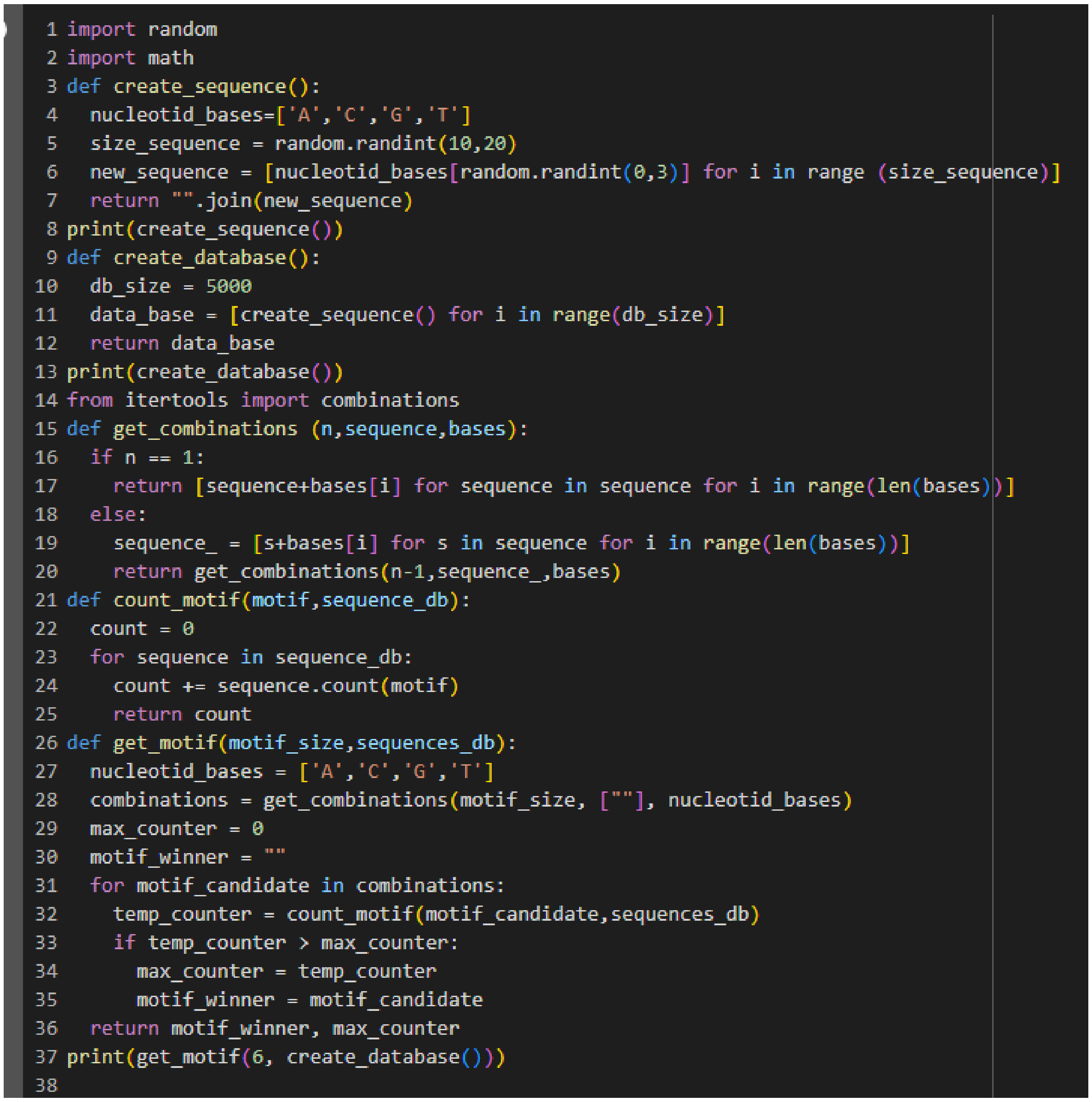
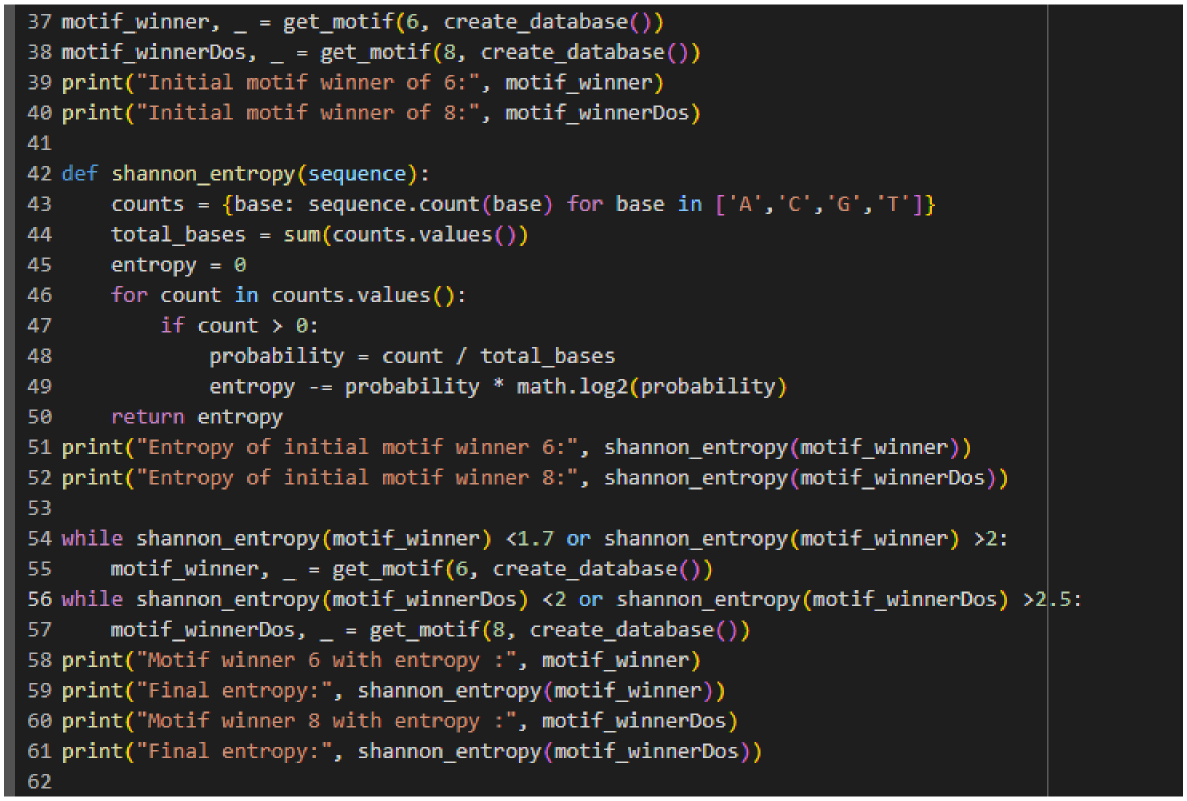
Algorithms



```
import random
import math
def create_sequence():
  nucleotid_bases=['A','C','G','T']
  size_sequence = random.randint(10,20)
  new_sequence = [nucleotid_bases[random.randint(0,3)] for i in range (size_sequence)]
  return "".join(new_sequence)
print(create_sequence())
def create_database():
  db_size = 5000
  data_base = [create_sequence() for i in range(db_size)]
  return data_base
print(create_database())
from itertools import combinations
def get_combinations (n,sequence,bases):
  if n == 1:
    return [sequence+bases[i] for sequence in sequence for i in range(len(bases))]
  else:
    sequence_ = [s+bases[i] for s in sequence for i in range(len(bases))]
    return get_combinations(n-1,sequence_,bases)
def count_motif(motif,sequence_db):
  count = 0
  for sequence in sequence_db:
    count += sequence.count(motif)
    return count
def get_motif(motif_size,sequences_db):
  nucleotid_bases = ['A','C','G','T']
  combinations = get_combinations(motif_size, [""], nucleotid_bases)
  max_counter = 0
  motif_winner = ""
  for motif_candidate in combinations:
    temp_counter = count_motif(motif_candidate,sequences_db)
    if temp_counter > max_counter:
      max_counter = temp_counter
      motif_winner = motif_candidate
  return motif_winner, max_counter
print(get_motif(6, create_database()))

```

First of all, the code that the professor made in class using recursion and control structures



```python
motif_winner, _ = get_motif(6, create_database())
motif_winnerDos, _ = get_motif(8, create_database())
print("Initial motif winner of 6:", motif_winner)
print("Initial motif winner of 8:", motif_winnerDos)

def shannon_entropy(sequence):
    counts = {base: sequence.count(base) for base in ['A','C','G','T']}
    total_bases = sum(counts.values())
    entropy = 0
    for count in counts.values():
        if count > 0:
            probability = count / total_bases
            entropy -= probability * math.log2(probability)
    return entropy
print("Entropy of initial motif winner 6:", shannon_entropy(motif_winner))
print("Entropy of initial motif winner 8:", shannon_entropy(motif_winnerDos))

while shannon_entropy(motif_winner) <1.7 or shannon_entropy(motif_winner) >2:
    motif_winner, _ = get_motif(6, create_database())
while shannon_entropy(motif_winnerDos) <2 or shannon_entropy(motif_winnerDos) >2.5:
    motif_winnerDos, _ = get_motif(8, create_database())
print("Motif winner 6 with entropy :", motif_winner)
print("Final entropy:", shannon_entropy(motif_winner))
print("Motif winner 8 with entropy :", motif_winnerDos)
print("Final entropy:", shannon_entropy(motif_winnerDos))

```

Lately I save the motif winner to use the string in the calculate of entropy and last I create a cycle that depends of the value of the entropy this will be changed meanwhile the strings change and will stop when the algorithm found a string whit a entropy between 1.7 and 2.

Conclusions

- If the value of the entropy is lower than 1 the string will have so many repeated bases or letters
- I define the maximum entropy in 2 because higher is so difficult to found a string with that entropy
- The value of the entropy changes depends of the size of the string