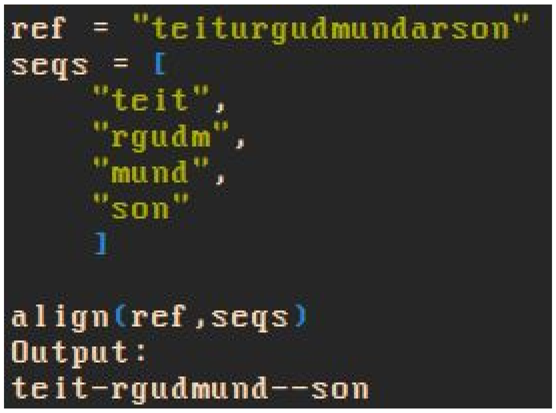
align(text,pattern) aligns the short sequences to the reference. It creates a string s with the same length as the reference string. It consists only with the letter x. It looks for entire sequences in the reference string. If it is found it is placed in s on the same index it appears in the reference. Sequences that are not found are ignored.
It works like so:



```
ref = "teiturgudmundarson"
seqs = [
    "teit",
    "rgudm",
    "mund",
    "son"
    ]

align(ref,seqs)
Output:
teit-rgudmund--son
```

Only we changed the dash later on to "x" because when checking how many unfilled spots there are, it takes python way longer to read a dash than a letter. Probably because in unicode a dash and most symbols are read as 2 or more characters.

We created aligned_reference.fasta with this function, to save time so we did not have to align it every time we ran the program.

get_missing_indices(aligned_reference) parses through the newly aligned reference and looks for location of indices where bases were missing. ***get_missing_indices_brief(aligned_reference)*** returns the indices without all the filler e.g. instead of 0,1, …, 38, it returns 0 and 38.

find_similar(mi_tuple, size) takes in the starting and ending index in the reference where bases are missing – this is where the brief version of get_missing_indices comes in handy – as well as size. It tries to pattern match in three ways (and uses the helper function ***iterate_seqs_with_re(pattern)*** for that purpose), so, for demonstration, if we have size 15 and:

ATGCATTTATTTACTTxxxxxxxxxCAATTATGTAATTTACCATGC

It searches for:

TGCATTTATTTACTT[ACGT]{9}
[ACGT]{9}CAATTATGTAATTTA
TGCATTTATTTACTT [ACGT]{9}CAATTATGTAATTTA

And returns a list of all the [ACGT]{9} matches.

get_consensus_sequence(matches) takes in all the matches from the previous sections and sums up, for each missing index, the most common base and splices it into a string, which it returns. For example, if we have the list:

```
TCTGGTAATC
ATTGGTCATC
ATTGGTCATC
AATCGTCATC
GTTGGTCATG
```

It returns:

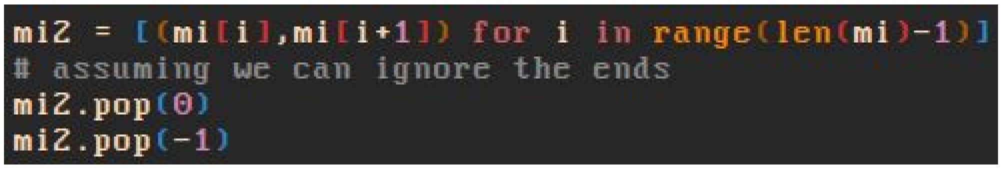
```
ATTGGTCATC
```

On the final day we received some hints from the teacher – the mutation at index 11083 turns from G to T, which matched our result.

We had some trouble filling in the head and tail.

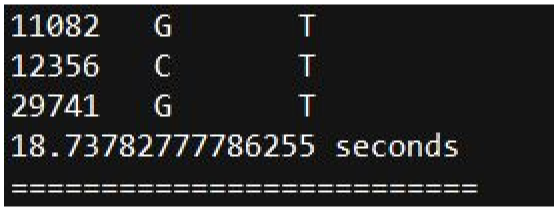
The teacher confirmed to us, that the bases would probably be missing at the ends.

Since we are working on a particular case and not developing general software, we decided to simply add the missing indices we were working with to an array as such:



```python
mi2 = [(mi[i],mi[i+1]) for i in range(len(mi)-1)]
# assuming we can ignore the ends
mi2.pop(0)
mi2.pop(-1)
```

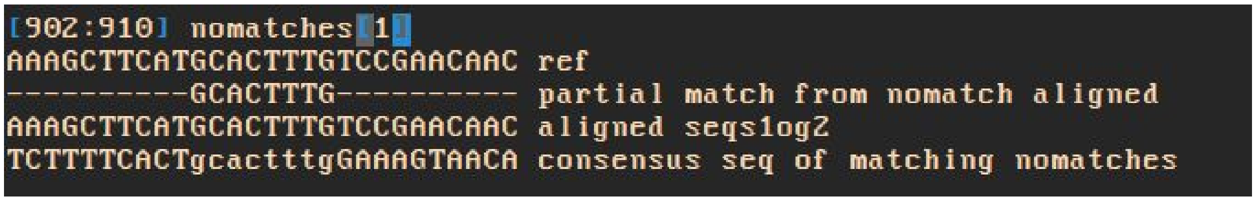
The teacher reaveled that there were about 8 mutations. We were disheartened to see that we found only three mutations.



```
11082   G       T
12356   C       T
29741   G       T
18.73782777786255 seconds
=========================
```

Clearly, somewhere we went astray. The fault may lie in the *align* function. It immediately plugs in entire matching sequences. There could be other sequences or sections of sequences that match almost exactly to the reference save for a small interval where a consensus of partially matching sequences might suggest a mutation. That would mean the matching sequence is actually a deviation and not a true match.

To test this we developed a new function:
def find_single_partial_pattern(ref,pattern): takes in the reference and a non-matching pattern from seqs. It find a substring of the sequence that *does* match the reference.
We used *align* with only that one pattern, returning an alignment reference with just that pattern inserted. We compared this to the reference and the fully aligned reference from earlier and looked at the letters around the partial matching pattern. We made a consensus string out all other non-matching sequences that matched this pattern.



```
[902:910] nomatches[1]
AAAGCTTCATGCACTTTGTCCGAACAAC ref
----------GCACTTTG---------- partial match from nomatch aligned
AAAGCTTCATGCACTTTGTCCGAACAAC aligned seqslog2
TCTTTTCACTgcactttgGAAAGTAACA consensus seq of matching nomatches
```

This suggest that our suspicions are true. That there are mutations that *align* ignores. However as we can see in the picture above there are far more than 8 mutations resulting from this consensus. We wanted to test all of the partial matches from non-mathes but running *find_single_partial_pattern* on just one non-matching sequence takes a whopping 16 seconds. Running all of them would take 30 days. So on the one hand we have too few mutations and on the other we have too many.
We could not conclude definitively what could be wrong in our algorithms.

Files:
sars_sequencing.py
Our solution. Takes about 26 seconds to run. While working on it we would only run align once in a separate program, print the result and output that to a separate fasta file to save time while testing the consensus process in a second program. The parsed sequences could also be outputted to a txt file that could then be read at the beginning of the second program to save extra time.

teitlib.py
Some helper functions, functions from the main program, and non-list versions of functions in main program.

output.txt
Final output file