Homework Exercise 11

1)

A)

Based on literature (Wikipedia is fine), what are the roles attributed to the PDZ domain?

B)

Find all UniProt keywords that are significantly associated with the **PDZ** domain. Should this analysis be carried out at the protein or domain level? Why?

What could explain the obtained significant associations? Write down an exhaustive list of explanations.

Which of the found keywords are compatible with the roles of PDZ you described in (A)? Which ones you wouldn't expect?

Bonus Questions

1)

Explain the following

i) *getattr* & *setattr* builtin functions
ii) *__getattr__*, *__setattr__* & *__getattribute__* special class functions
iii) *locals* & *globals* builtin functions

Demonstrate *__getattr__* by writing a class called *MotifCounter* that counts the number of occurrences of given DNA motifs within a given DNA sequence. *MotifCounter* should support the following syntax:

 # Initializing a MotifCounter with a given DNA sequence

>> motif_counter = MotifCounter('ACGTTGGACAGTGACCAGTGGGAATGCGTTTATAGCCAGGAT')

 # Count the number of occurrences of the GGA motif

>> print(motif_counter.GGA)

3

(Of course GGA is arbitrary; the class should support any motif.)

2)

A new B.Sc. project student from your lab was tasked to analyze kmer distribution (i.e. the frequencies of all possible nucleotide sequences of length k) in the *E. coli* genome. He wrote a script that collected, for each kmer of length 4, the indexes where that kmer appears in the genome. However, the script runs for so long it's impractical to use. He asked for your help.

1. Explain what the term "profiling" (in the context of programming).
2. Use (and explain how to run) the line_profiler tool on the student's code (you will find it in the **"Profiling question handout"** folder).

   Instructions: running the profiler on code that takes a long time to finish is not a good strategy, as running a script with a profiler attached will take even longer! Instead, run it on a small test input, or limit the number of kmers that the script will try to find.
3. Show and explain the output of the program from step 2.

   Some of the frequent performance issues that tend to happen are:
   i) Input/output - reading or writing the data (and sometimes storing it in memory) may take more time than anything else in the code

ii) Manual vs. built-in implementation - especially in high-level languages like Python, many basic algorithms implemented as built-ins (e.g., as part of data structures and modules that come with the language) will be much faster than what we can write. For example, Python's list.sort() and sorted() will run much faster than any sorting algorithm we will implement ourselves. For code that runs for a long time, it may be worthwhile to consider if it can be changed so as to leverage language built-ins rather than pure Python code.

iii) Algorithmic mistakes - sometimes the logic of the algorithm is inherently slow. For example, nested loops where we can only use a single loop, looking up a value in a list with a loop instead of using a dictionary lookup, etc. These often give the best improvements, but also require thought and experience.

However, it's hard to know in advance what the problems in your script are - this is why a profiler should be used!

4. What are the performance issues in the script? Explain and show how to solve them. Can you find the locations of all 4-mers in the provided *E. coli* genome in under a minute?

   Instructions: even if your final version ends up not using the find_kmer() function, you should still try to optimize it and explain how you did so.