Exam project for Applied Programming 2011: Truncated coding reading frames.

Imagine that you suspect that a type of cancer is caused by/correlated with the malfunctioning of one of a number of candidate genes. Your hypothesis is that the protein product of a gene is truncated as a result of a somatic mutation introducing a stop codon (TGA, TAG or TAA). You want to screen for such truncated proteins in samples of cancer cells. To do this you need to know which truncated products to screen for. The task at hand is to write code that lets you identify at what positions in the coding reading frame (CDS) of each gene that a single somatic mutation may introduce a stop codon. Having done this you would also like some general statistics across all analyzed genes about the codons you identify.

Along with this assignment there is also a file (EksamensOpgaveAP2011inputData.py) with example input data that can be downloaded from where you also found this file.

Problem 1:
The first task is to parse the CDS annotation for a sequence so we know which parts code for protein.

Write a function

```
def parseAnnotation(annotation):
```

that takes a string `annotation` as argument. This string describes the location of the CDS in the sequence in the following manner (start..end),(start..end) etc. E.g. the string `'(459..521),(1834..2736)'` specifies that the CDS is distributed in two exons with one part from base number 459 to base number 521 (both inclusive) and the remaining part from base number 1834 to base number 2736 (both inclusive). The function must return a list with a tuple for each CDS part, each with a start and end value.

Example usage:
```
parseAnnotation('(459..521),(1834..2736)')
```

should return
```
[(459, 521), (1834, 2736)]
```

Problem 2:
Next thing is to extract the CDS from the sequence.

Write a function

```
def cutoutCDS(seq, annotation):
```

that takes a string `seq` containing a DNA sequence and a string `annotation` containing an annotation string. The function must return a string containing the CDS sequence.

Problem 3:
To analyze CDSs we need to be able to evaluate the differences between two codons.

Write a function:

```
def differencesBetweenCodons(codon1, codon2):
```

that takes two string arguments, `codon1` and `codon2`, each containing a codon (three nucleotides). The function must return a list of length three of zeros and ones. Each element in this list must indicate if the two codons are different at that position. That is, if the first bases in each codon are different the first element in the list is `1` if they are the same it must be `0`.

Example usage:
```
differencesBetweenCodons("TAG", "GAT")
```

should return
```
[1,0,1]
```

because the first and the last nucleotides differ.

Problem 4:
Now that we can locate the differences between two codons, we can go on to implement a function that computes how close a codon is to being a stop codon. The function we need should return the stop codon that has the fewest differences to `codon` but also information about how much they are different and how they are different.

Write a function:

```
def differencesToStopCodon(codon):
```

that takes a string of three characters `codon` as argument. The function must return three values in the following order: a string containing the stop codon that `codon` is most similar to, an integer containing the number of differences between `codon` and the returned stop codon, and a list with three zeros or ones (as described in problem 3) representing at which positions `codon` differs from the returned stop codon.

Example usage:
```
differencesToStopCodon("GAA")
```

should return
`("TAA", 1, [1,0,0])`

Problem 5:
Next step is to identify all positions in the CDS (except of cause the stop codon) where *one* mutation may turn a codon into a stop codon

Write a function

```
def findPotentialStopCodons(cds):
```

that takes a string argument `cds` containing a CDS sequence. The function must search all non-overlapping codons and return a list of tuples with a tuple for each codon that may mutate into a stop codon. Each tuple must contain: a codon that may mutate into a stop codon, the stop codon this would mutate into, and the base number in the CDS where a mutation will cause this change.

Example usage:
`findPotentialStopCodons('ATGCAATGTTAA')`

should return
`[('CAA', 'TAA', 4), ('TGT', 'TGA', 9)]`

Now that you can find potential stop codons you can easily compute the resulting truncated proteins. We did this in the course so we don't need to to take that any further here.

Problem 6:
Lets say you have analyzed a number of genes using the code you have written. In addition to finding out where potential stop codons may be introduced in each particular CDSs, you are also interested in whether there are any general trends in how stop codons may be introduced. To be able to summarize your results for multiple CDSs you must store the results for all the genes you analyze in a single data structure.

You must use a dictionary of dictionaries for this purpose. Do this in a way so that you can access a list of all observed base positions where a `stopcodon` has mutated from a `codon` in this way: `d[stopcodon][codon]`. To do this you must write a function that adds the information returned by `findPotentialStopCodons` to such a dictionary.

Write a function

```
def addToMutationDictionary(d, tup):
```

that takes a dictionary argument `d` and a tuple argument `tup`.

The dictionary `d` may be empty or may already contain data as described above. As explained in problem 5, the tuple `tup` contains the codon, the potential stop codon, and an integer describing a base position in the CDS where a mutation will cause the change. The function must add the data in `tup` to the data structure `d`. The function must not return anything (other than `None` of course).

Problem 7:
Now you must produce a summarized account of all your observations across all CDSs. Calculate the frequencies of the codons you have observed may mutate into a stop codon. You must group your observations by stop codon so that frequencies for each stop codon sum to one.

Write a function:

```
def printStatistics(cdsList):
```

that takes a list argument `cdsList` containing CDS sequences. The function should print a summary of the codons observed as described above. The output must be formatted exactly as in the example below with one tab for indentation and two decimals on the frequencies.

Example usage:

```
printStatistics(['ATGCAAAAGTAA', 'ATGTGCAGATGA'])
```

should print:

```
TAG:
     AAG: 1.00
TGA:
     TGG: 0.50
     AGA: 0.50
TAA:
     CAA: 1.00
```

Problem 8:
Now that you can compute the CDS positions where mutations may introduce stop codons you also need to be able to find the corresponding positions in the un-spliced sequence with introns included.

Write a function

```
def convertToSequenceCoordinates(pos, annotation):
```

that takes an integer argument `pos` containing the CDS base position and a string argument `annotation` that contains CDS annotation for the sequence. The function must return the position in the un-spliced sequence corresponding to the CDS position given as argument.