# This is CS50x

## DNA

Implement a program that identifies a person based on their DNA, per the below.

```
$ python dna.py databases/large.csv sequences/5.txt
Lavender
```

## Getting Started

Here's how to download this problem into your own CS50 IDE. Log into CS50 IDE (https://ide.cs50.io/) and then, in a terminal window, execute each of the below.

- Navigate to your `pset6` directory that should already exist.
- Execute `wget https://cdn.cs50.net/2020/fall/psets/6/dna/dna.zip` to download a (compressed) ZIP file with this problem's distribution.
- Execute `unzip dna.zip` to uncompress that file.
- Execute `rm dna.zip` followed by `yes` or `y` to delete that ZIP file.
- Execute `ls`. You should see a directory called `dna`, which was inside of that ZIP file.
- Execute `cd dna` to change into that directory.
- Execute `ls`. You should see a directory of sample `databases` and a directory of sample `sequences`.

## Background

DNA, the carrier of genetic information in living things, has been used in criminal justice for decades. But how, exactly, does DNA profiling work? Given a sequence of DNA, how can forensic investigators identify to whom it belongs?

Well, DNA is really just a sequence of molecules called nucleotides, arranged into a particular shape (a double helix). Each nucleotide of DNA contains one of four different bases: adenine (A), cytosine (C), guanine (G), or thymine (T). Every human cell has billions of these nucleotides arranged in sequence. Some portions of this sequence (i.e. genome) are the same, or at least very similar, across almost all humans, but other portions of the sequence have a higher genetic diversity and thus vary more across the population.

One place where DNA tends to have high genetic diversity is in Short Tandem Repeats (STRs). An STR is a short sequence of DNA bases that tends to repeat consecutively numerous times at specific locations inside of a person's DNA. The number of times any particular STR repeats varies a lot among individuals. In the DNA samples below, for example, Alice has the STR `AGAT` repeated four times in her DNA, while Bob has the same STR repeated five times.

# Alice: CT⌈AGAT⌉⌈AGAT⌉⌈AGAT⌉⌈AGAT⌉GACTA

# Bob: CT⌈AGAT⌉⌈AGAT⌉⌈AGAT⌉⌈AGAT⌉⌈AGAT⌉T

Using multiple STRs, rather than just one, can improve the accuracy of DNA profiling. If the probability that two people have the same number of repeats for a single STR is 5%, and the analyst looks at 10 different STRs, then the probability that two DNA samples match purely by chance is about 1 in 1 quadrillion (assuming all STRs are independent of each other). So if two DNA samples match in the number of repeats for each of the STRs, the analyst can be pretty confident they came from the same person. CODIS, The FBI's [DNA database (https://www.fbi.gov/services/laboratory/biometric-analysis/codis/codis-and-ndis-fact-sheet)](https://www.fbi.gov/services/laboratory/biometric-analysis/codis/codis-and-ndis-fact-sheet), uses 20 different STRs as part of its DNA profiling process.

What might such a DNA database look like? Well, in its simplest form, you could imagine formatting a DNA database as a CSV file, wherein each row corresponds to an individual, and each column corresponds to a particular STR.

```
name,AGAT,AATG,TATC
Alice,28,42,14
Bob,17,22,19
Charlie,36,18,25
```

The data in the above file would suggest that Alice has the sequence `AGAT` repeated 28 times consecutively somewhere in her DNA, the sequence `AATG` repeated 42 times, and `TATC` repeated 14 times. Bob, meanwhile, has those same three STRs repeated 17 times, 22 times, and 19 times, respectively. And Charlie has those same three STRs repeated 36, 18, and 25 times, respectively.

So given a sequence of DNA, how might you identify to whom it belongs? Well, imagine that you looked through the DNA sequence for the longest consecutive sequence of repeated `AGAT` s and found that the longest sequence was 17 repeats long. If you then found that the longest sequence of `AATG` is 22 repeats long, and the longest sequence of `TATC` is 19 repeats long, that would provide pretty good evidence that the DNA was Bob's. Of course, it's also possible that once you take the counts for each of the STRs, it doesn't match anyone in your DNA database, in which case you have no match.

In practice, since analysts know on which chromosome and at which location in the DNA an STR will be found, they can localize their search to just a narrow section of DNA. But we'll ignore that detail for this problem.

Your task is to write a program that will take a sequence of DNA and a CSV file containing STR counts for a list of individuals and then output to whom the DNA (most likely) belongs.

## Specification

In a file called `dna.py` in `~/pset6/dna/`, implement a program that identifies to whom a sequence of DNA belongs.

- The program should require as its first command-line argument the name of a CSV file containing the STR counts for a list of individuals and should require as its second command-line argument the name of a text file containing the DNA sequence to identify.
  - If your program is executed with the incorrect number of command-line arguments, your program should print an error message of your choice (with `print`). If the correct number of arguments are provided, you may assume that the first argument is indeed the filename of a valid CSV file, and that the second argument is the filename of a valid text file.
- Your program should open the CSV file and read its contents into memory.
  - You may assume that the first row of the CSV file will be the column names. The first column will be the word `name` and the remaining columns will be the STR sequences themselves.
- Your program should open the DNA sequence and read its contents into memory.
- For each of the STRs (from the first line of the CSV file), your program should compute the longest run of consecutive repeats of the STR in the DNA sequence to identify.
- If the STR counts match exactly with any of the individuals in the CSV file, your program should print out the name of the matching individual.
  - You may assume that the STR counts will not match more than one individual.
  - If the STR counts do not match exactly with any of the individuals in the CSV file, your program should print `"No match"`.

## Walkthrough

## Usage

Your program should behave per the example below:

```
$ python dna.py databases/large.csv sequences/5.txt
Lavender
```

```
$ python dna.py
Usage: python dna.py data.csv sequence.txt
```

```
$ python dna.py data.csv
Usage: python dna.py data.csv sequence.txt
```

## Hints

- You may find Python's `csv` (https://docs.python.org/3/library/csv.html) module helpful for reading CSV files into memory. You may want to take advantage of either `csv.reader` (https://docs.python.org/3/library/csv.html#csv.reader) or `csv.DictReader` (https://docs.python.org/3/library/csv.html#csv.DictReader).
- The `open` (https://docs.python.org/3.3/tutorial/inputoutput.html#reading-and-writing-files) and `read` (https://docs.python.org/3.3/tutorial/inputoutput.html#methods-of-file-objects) functions may prove useful for reading text files into memory.
- Consider what data structures might be helpful for keeping tracking of information in your program. A `list` (https://docs.python.org/3/tutorial/introduction.html#lists) or a `dict` (https://docs.python.org/3/tutorial/datastructures.html#dictionaries) may prove useful.
- Python strings allow "slicing" (accessing a particular substring within a string). If `s` is a string, then `s[i:j]` will return a new string with just the characters of `s` starting from character `i` up through (but not including) character `j`.
- It may be helpful to start by writing a function that, given both a DNA sequence and an STR as inputs, returns the maximum number of times that the STR repeats. You can then use that function in other parts of your program!

## Testing

While `check50` is available for this problem, you're encouraged to first test your code on your own for each of the following.

- Run your program as `python dna.py databases/small.csv sequences/1.txt`. Your program should output `Bob`.
- Run your program as `python dna.py databases/small.csv sequences/2.txt`. Your program should output `No match`.
- Run your program as `python dna.py databases/small.csv sequences/3.txt`. Your program should output `No match`.
- Run your program as `python dna.py databases/small.csv sequences/4.txt`. Your program should output `Alice`.
- Run your program as `python dna.py databases/large.csv sequences/5.txt`. Your program should output `Lavender`.
- Run your program as `python dna.py databases/large.csv sequences/6.txt`. Your program should output `Luna`.
- Run your program as `python dna.py databases/large.csv sequences/7.txt`. Your program should output `Ron`.
- Run your program as `python dna.py databases/large.csv sequences/8.txt`. Your program should output `Ginny`.
- Run your program as `python dna.py databases/large.csv sequences/9.txt`. Your program should output `Draco`.
- Run your program as `python dna.py databases/large.csv sequences/10.txt`. Your program should output `Albus`.

- Run your program as `python dna.py databases/large.csv sequences/11.txt`. Your program should output `Hermione`.
- Run your program as `python dna.py databases/large.csv sequences/12.txt`. Your program should output `Lily`.
- Run your program as `python dna.py databases/large.csv sequences/13.txt`. Your program should output `No match`.
- Run your program as `python dna.py databases/large.csv sequences/14.txt`. Your program should output `Severus`.
- Run your program as `python dna.py databases/large.csv sequences/15.txt`. Your program should output `Sirius`.
- Run your program as `python dna.py databases/large.csv sequences/16.txt`. Your program should output `No match`.
- Run your program as `python dna.py databases/large.csv sequences/17.txt`. Your program should output `Harry`.
- Run your program as `python dna.py databases/large.csv sequences/18.txt`. Your program should output `No match`.
- Run your program as `python dna.py databases/large.csv sequences/19.txt`. Your program should output `Fred`.
- Run your program as `python dna.py databases/large.csv sequences/20.txt`. Your program should output `No match`.

Execute the below to evaluate the correctness of your code using `check50`. But be sure to compile and test it yourself as well!

```
check50 cs50/problems/2021/x/dna
```

Execute the below to evaluate the style of your code using `style50`.

```
style50 dna.py
```

## How to Submit

Execute the below, logging in with your GitHub username and password when prompted. For security, you'll see asterisks (`*`) instead of the actual characters in your password.

```
submit50 cs50/problems/2021/x/dna
```