

This is CS50x

OpenCourseWare

David J. Malan (<https://cs.harvard.edu/malan/>)

malan@harvard.edu

[f](https://www.facebook.com/dmalan/) (<https://www.facebook.com/dmalan/>) [G](https://github.com/dmalan/) (<https://github.com/dmalan/>) [@](https://www.instagram.com/davidjmalan/) (<https://www.instagram.com/davidjmalan/>) [in](https://www.linkedin.com/in/malan/)

(<https://www.linkedin.com/in/malan/>) [Q](https://www.quora.com/profile/David-J-Malan) (<https://www.quora.com/profile/David-J-Malan>) [T](https://twitter.com/davidjmalan/) (<https://twitter.com/davidjmalan/>)

Problem Set 6

What to Do

1. Submit [Hello](#) in Python
2. Submit one of:
 - [this version of Mario](#) in Python, if feeling less comfortable
 - [this version of Mario](#) in Python, if feeling more comfortable
3. Submit one of:
 - [Cash](#) in Python, if feeling less comfortable
 - [Credit](#) in Python, if feeling more comfortable
4. Submit [Readability](#) in Python
5. Submit [DNA](#) in Python

If you submit both versions of Mario, we'll record the higher of your two scores. If you submit both Cash and Credit, we'll record the higher of your two scores.

When to Do It

By 11:59pm on 31 December 2020.

Advice

- Try out any of David's programs from class via [Week 6's](#) source code.

Academic Honesty

- For Hello, Mario, Cash, Credit, and Readability, it is **reasonable** to look at your own implementations thereof in C and others' implementations thereof *in C*, including the staff's implementations thereof in C.
- It is **not reasonable** to look at others' implementations of the same *in Python*.
- Insofar as a goal of these problems is to teach you how to teach yourself a new language, keep in mind that these acts are not only **reasonable**, per the syllabus, but encouraged toward that end:
 - Incorporating a few lines of code that you find online or elsewhere into your own code, provided that those lines are not themselves solutions to assigned problems and that you cite the lines' origins.
 - Turning to the web or elsewhere for instruction beyond the course's own, for references, and for solutions to technical difficulties, but not for outright solutions to problem set's problems or your own final project.

This is CS50x

OpenCourseWare

David J. Malan (<https://cs.harvard.edu/malan/>)

malan@harvard.edu

[f](https://www.facebook.com/dmalan/) (<https://www.facebook.com/dmalan/>) [G](https://github.com/dmalan/) (<https://github.com/dmalan/>) [@](https://www.instagram.com/davidjmalan/) (<https://www.instagram.com/davidjmalan/>) [in](https://www.linkedin.com/in/malan/)

(<https://www.linkedin.com/in/malan/>) [Q](https://www.quora.com/profile/David-J-Malan) (<https://www.quora.com/profile/David-J-Malan>) [T](https://twitter.com/davidjmalan/) (<https://twitter.com/davidjmalan/>)

Hello

Implement a program that prints out a simple greeting to the user, per the below.

```
$ python hello.py
What is your name?
David
hello, David
```

Specification

Write, in a file called `hello.py` in `~/pset6/hello`, a program that prompts a user for their name, and then prints `hello, so-and-so`, where `so-and-so` is their provided name, exactly as you did in [Problem Set 1](#), except that your program this time should be written (a) in Python and (b) in CS50 IDE.

Usage

Your program should behave per the example below.

```
$ python hello.py
What is your name?
Emma
hello, Emma
```

Testing

No `check50` for this problem, but be sure to test your code for each of the following.

- Run your program as `python hello.py`, and wait for a prompt for input. Type in `Emma` and press enter. Your program should output `hello, Emma`.
- Run your program as `python hello.py`, and wait for a prompt for input. Type in `Rodrigo` and press enter. Your program should output `hello, Rodrigo`.

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/2020/x/sentimental/hello
```


This is CS50x

OpenCourseWare

David J. Malan (<https://cs.harvard.edu/malan/>)

malan@harvard.edu

[f](https://www.facebook.com/dmalan/) (<https://www.facebook.com/dmalan/>) [G](https://github.com/dmalan/) (<https://github.com/dmalan/>) [@](https://www.instagram.com/davidjmalan/) (<https://www.instagram.com/davidjmalan/>) [in](https://www.linkedin.com/in/malan/)

(<https://www.linkedin.com/in/malan/>) [Q](https://www.quora.com/profile/David-J-Malan) (<https://www.quora.com/profile/David-J-Malan>) [T](https://twitter.com/davidjmalan) (<https://twitter.com/davidjmalan>)

Mario



Implement a program that prints out a half-pyramid of a specified height, per the below.

```
$ ./mario
Height: 4
#
##
###
####
```

Specification

- Write, in a file called `mario.py` in `~/pset6/mario/less/`, a program that recreates the half-pyramid using hashes (`#`) for blocks, exactly as you did in [Problem Set 1](#), except that your program this time should be written (a) in Python and (b) in CS50 IDE.
- To make things more interesting, first prompt the user with `get_int` for the half-pyramid's height, a positive integer between `1` and `8`, inclusive.
- If the user fails to provide a positive integer no greater than `8`, you should re-prompt for the same again.
- Then, generate (with the help of `print` and one or more loops) the desired half-pyramid.
- Take care to align the bottom-left corner of your half-pyramid with the left-hand edge of your terminal window.

Usage

Your program should behave per the example below.

```
$ ./mario
Height: 4
#
##
###
####
```

Testing

No `check50` for this problem, but be sure to test your code for each of the following

NO `check50` for this problem, but be sure to test your code for each of the following.

- Run your program as `python mario.py` and wait for a prompt for input. Type in `-1` and press enter. Your program should reject this input as invalid, as by re-prompting the user to type in another number.
- Run your program as `python mario.py` and wait for a prompt for input. Type in `0` and press enter. Your program should reject this input as invalid, as by re-prompting the user to type in another number.
- Run your program as `python mario.py` and wait for a prompt for input. Type in `1` and press enter. Your program should generate the below output. Be sure that the pyramid is aligned to the bottom-left corner of your terminal, and that there are no extra spaces at the end of each line.

```
#
```

- Run your program as `python mario.py` and wait for a prompt for input. Type in `2` and press enter. Your program should generate the below output. Be sure that the pyramid is aligned to the bottom-left corner of your terminal, and that there are no extra spaces at the end of each line.

```
#  
##
```

- Run your program as `python mario.py` and wait for a prompt for input. Type in `8` and press enter. Your program should generate the below output. Be sure that the pyramid is aligned to the bottom-left corner of your terminal, and that there are no extra spaces at the end of each line.

```
      #  
     ##  
    ###  
   ####  
  #####  
 #####  
#####  
#####  
#####
```

- Run your program as `python mario.py` and wait for a prompt for input. Type in `9` and press enter. Your program should reject this input as invalid, as by re-prompting the user to type in another number. Then, type in `2` and press enter. Your program should generate the below output. Be sure that the pyramid is aligned to the bottom-left corner of your terminal, and that there are no extra spaces at the end of each line.

```
#  
##
```

- Run your program as `python mario.py` and wait for a prompt for input. Type in `foo` and press enter. Your program should reject this input as invalid, as by re-prompting the user to type in another number.
- Run your program as `python mario.py` and wait for a prompt for input. Do not type anything, and press enter. Your program should reject this input as invalid, as by re-prompting the user to type in another number.

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/2020/x/sentimental/mario/less
```


This is CS50x

OpenCourseWare

David J. Malan (<https://cs.harvard.edu/malan/>)

malan@harvard.edu

[f](https://www.facebook.com/dmalan/) (<https://www.facebook.com/dmalan/>) [G](https://github.com/dmalan/) (<https://github.com/dmalan/>) [@](https://www.instagram.com/davidjmalan/) (<https://www.instagram.com/davidjmalan/>) [in](https://www.linkedin.com/in/malan/)

(<https://www.linkedin.com/in/malan/>) [Q](https://www.quora.com/profile/David-J-Malan) (<https://www.quora.com/profile/David-J-Malan>) [T](https://twitter.com/davidjmalan/) (<https://twitter.com/davidjmalan/>)

Mario



Implement a program that prints out a double half-pyramid of a specified height, per the below.

```
$ ./mario
Height: 4
# #
## ##
### ###
#### ####
```

Specification

- Write, in a file called `mario.py` in `~/pset6/mario/more/`, a program that recreates these half-pyramids using hashes (`#`) for blocks, exactly as you did in [Problem Set 1](#), except that your program this time should be written (a) in Python and (b) in CS50 IDE.
- To make things more interesting, first prompt the user with `get_int` for the half-pyramid's height, a positive integer between `1` and `8`, inclusive. (The height of the half-pyramids pictured above happens to be `4`, the width of each half-pyramid `4`, with a gap of size `2` separating them).
- If the user fails to provide a positive integer no greater than `8`, you should re-prompt for the same again.
- Then, generate (with the help of `print` and one or more loops) the desired half-pyramids.
- Take care to align the bottom-left corner of your pyramid with the left-hand edge of your terminal window, and ensure that there are two spaces between the two pyramids, and that there are no additional spaces after the last set of hashes on each row.

Usage

Your program should behave per the example below.

```
$ ./mario
Height: 4
# #
## ##
### ###
#### ####
```

Testing

No `check50` for this problem, but be sure to test your code for each of the following.

- Run your program as `python mario.py` and wait for a prompt for input. Type in `-1` and press enter. Your program should reject this input.

- Run your program as `python mario.py` and wait for a prompt for input. Type in `-1` and press enter. Your program should reject this input as invalid, as by re-prompting the user to type in another number.
- Run your program as `python mario.py` and wait for a prompt for input. Type in `0` and press enter. Your program should reject this input as invalid, as by re-prompting the user to type in another number.
- Run your program as `python mario.py` and wait for a prompt for input. Type in `1` and press enter. Your program should generate the below output. Be sure that the pyramid is aligned to the bottom-left corner of your terminal, and that there are no extra spaces at the end of each line.

```
# #
```

- Run your program as `python mario.py` and wait for a prompt for input. Type in `2` and press enter. Your program should generate the below output. Be sure that the pyramid is aligned to the bottom-left corner of your terminal, and that there are no extra spaces at the end of each line.

```
# #
## ##
```

- Run your program as `python mario.py` and wait for a prompt for input. Type in `8` and press enter. Your program should generate the below output. Be sure that the pyramid is aligned to the bottom-left corner of your terminal, and that there are no extra spaces at the end of each line.

```
  # #
 ## ##
### ###
#### ####
##### #####
##### #####
##### #####
##### #####
##### #####
```

- Run your program as `python mario.py` and wait for a prompt for input. Type in `9` and press enter. Your program should reject this input as invalid, as by re-prompting the user to type in another number. Then, type in `2` and press enter. Your program should generate the below output. Be sure that the pyramid is aligned to the bottom-left corner of your terminal, and that there are no extra spaces at the end of each line.

```
# #
## ##
```

- Run your program as `python mario.py` and wait for a prompt for input. Type in `foo` and press enter. Your program should reject this input as invalid, as by re-prompting the user to type in another number.
- Run your program as `python mario.py` and wait for a prompt for input. Do not type anything, and press enter. Your program should reject this input as invalid, as by re-prompting the user to type in another number.

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/2020/x/sentimental/mario/more
```


This is CS50x

OpenCourseWare

David J. Malan (<https://cs.harvard.edu/malan/>)

malan@harvard.edu

[f](https://www.facebook.com/dmalan/) (<https://www.facebook.com/dmalan/>) [G](https://github.com/dmalan/) (<https://github.com/dmalan/>) [@](https://www.instagram.com/davidjmalan/) (<https://www.instagram.com/davidjmalan/>) [in](https://www.linkedin.com/in/malan/)

(<https://www.linkedin.com/in/malan/>) [Q](https://www.quora.com/profile/David-J-Malan) (<https://www.quora.com/profile/David-J-Malan>) [T](https://twitter.com/davidjmalan/) (<https://twitter.com/davidjmalan/>)

Cash

Implement a program that calculates the minimum number of coins required to give a user change.

```
$ python cash.py
Change owed: 0.41
4
```

Specification

- Write, in a file called `cash.py` in `~/pset6/cash/`, a program that first asks the user how much change is owed and then spits out the minimum number of coins with which said change can be made, exactly as you did in [Problem Set 1](#), except that your program this time should be written (a) in Python and (b) in CS50 IDE.
- Use `get_float` from the CS50 Library to get the user's input and `print` to output your answer. Assume that the only coins available are quarters (25¢), dimes (10¢), nickels (5¢), and pennies (1¢).
 - We ask that you use `get_float` so that you can handle dollars and cents, albeit sans dollar sign. In other words, if some customer is owed \$9.75 (as in the case where a newspaper costs 25¢ but the customer pays with a \$10 bill), assume that your program's input will be `9.75` and not `$9.75` or `975`. However, if some customer is owed \$9 exactly, assume that your program's input will be `9.00` or just `9` but, again, not `$9` or `900`. Of course, by nature of floating-point values, your program will likely work with inputs like `9.0` and `9.000` as well; you need not worry about checking whether the user's input is "formatted" like money should be.
- If the user fails to provide a non-negative value, your program should re-prompt the user for a valid amount again and again until the user complies.
- Incidentally, so that we can automate some tests of your code, we ask that your program's last line of output be only the minimum number of coins possible: an integer followed by a newline.

Usage

Your program should behave per the example below.

```
$ python cash.py
Change owed: 0.41
4
```

Testing

No `check50` for this problem, but be sure to test your code for each of the following.

- Run your program as `python cash.py`, and wait for a prompt for input. Type in `0.41` and press enter. Your program should output `4`.
- Run your program as `python cash.py`, and wait for a prompt for input. Type in `0.01` and press enter. Your program should output `1`.
- Run your program as `python cash.py`, and wait for a prompt for input. Type in `0.15` and press enter. Your program should output `2`.
- Run your program as `python cash.py`, and wait for a prompt for input. Type in `1.60` and press enter. Your program should output `7`.
- Run your program as `python cash.py`, and wait for a prompt for input. Type in `23` and press enter. Your program should output `92`.
- Run your program as `python cash.py`, and wait for a prompt for input. Type in `4.2` and press enter. Your program should output `18`.
- Run your program as `python cash.py`, and wait for a prompt for input. Type in `-1` and press enter. Your program should reject this input

as invalid, as by re-prompting the user to type in another number.

- Run your program as `python cash.py`, and wait for a prompt for input. Type in `foo` and press enter. Your program should reject this input as invalid, as by re-prompting the user to type in another number.
- Run your program as `python cash.py`, and wait for a prompt for input. Do not type anything, and press enter. Your program should reject this input as invalid, as by re-prompting the user to type in another number.

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/2020/x/sentimental/cash
```

This is CS50x

OpenCourseWare

David J. Malan (<https://cs.harvard.edu/malan/>)

malan@harvard.edu

[f](https://www.facebook.com/dmalan/) (<https://www.facebook.com/dmalan/>) [G](https://github.com/dmalan/) (<https://github.com/dmalan/>) [@](https://www.instagram.com/davidjmalan/) (<https://www.instagram.com/davidjmalan/>) [in](https://www.linkedin.com/in/malan/)

(<https://www.linkedin.com/in/malan/>) [Q](https://www.quora.com/profile/David-J-Malan/) (<https://www.quora.com/profile/David-J-Malan/>) [T](https://twitter.com/davidjmalan/) (<https://twitter.com/davidjmalan/>)

Credit

Implement a program that determines whether a provided credit card number is valid according to Luhn's algorithm.

```
$ python credit.py
Number: 378282246310005
AMEX
```

Specification

- In `credit.py` in `~/pset6/credit/`, write a program that prompts the user for a credit card number and then reports (via `print`) whether it is a valid American Express, MasterCard, or Visa card number, exactly as you did in [Problem Set 1](#), except that your program this time should be written (a) in Python and (b) in CS50 IDE.
- So that we can automate some tests of your code, we ask that your program's last line of output be `AMEX\n` or `MASTERCARD\n` or `VISA\n` or `INVALID\n`, nothing more, nothing less.
- For simplicity, you may assume that the user's input will be entirely numeric (i.e., devoid of hyphens, as might be printed on an actual card).
- Best to use `get_int` or `get_string` from CS50's library to get users' input, depending on how you to decide to implement this one.

Usage

Your program should behave per the example below.

```
$ python credit.py
Number: 378282246310005
AMEX
```

Testing

No `check50` for this problem, but be sure to test your code for each of the following.

- Run your program as `python credit.py`, and wait for a prompt for input. Type in `378282246310005` and press enter. Your program should output `AMEX`.
- Run your program as `python credit.py`, and wait for a prompt for input. Type in `371449635398431` and press enter. Your program should output `AMEX`.
- Run your program as `python credit.py`, and wait for a prompt for input. Type in `5555555555554444` and press enter. Your program should output `MASTERCARD`.
- Run your program as `python credit.py`, and wait for a prompt for input. Type in `5105105105105100` and press enter. Your program should output `MASTERCARD`.
- Run your program as `python credit.py`, and wait for a prompt for input. Type in `4111111111111111` and press enter. Your program should output `VISA`.
- Run your program as `python credit.py`, and wait for a prompt for input. Type in `4012888888881881` and press enter. Your program should output `VISA`.
- Run your program as `python credit.py`, and wait for a prompt for input. Type in `1234567890` and press enter. Your program should output `INVALID`.

Output: **INVALID**.

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/2020/x/sentimental/credit
```

This is CS50x

OpenCourseWare

David J. Malan (<https://cs.harvard.edu/malan/>)

malan@harvard.edu

[f](https://www.facebook.com/dmalan/) (<https://www.facebook.com/dmalan/>) [G](https://github.com/dmalan/) (<https://github.com/dmalan/>) [@](https://www.instagram.com/davidjmalan/) (<https://www.instagram.com/davidjmalan/>) [in](https://www.linkedin.com/in/malan/)

(<https://www.linkedin.com/in/malan/>) [Q](https://www.quora.com/profile/David-J-Malan) (<https://www.quora.com/profile/David-J-Malan>) [T](https://twitter.com/davidjmalan/) (<https://twitter.com/davidjmalan/>)

Readability

Implement a program that computes the approximate grade level needed to comprehend some text, per the below.

```
$ python readability.py
Text: Congratulations! Today is your day. You're off to Great Places! You're off and away!
Grade 3
```

Specification

- Write, in a file called `readability.py` in `~/pset6/readability/`, a program that first asks the user to type in some text, and then outputs the grade level for the text, according to the Coleman-Liau formula, exactly as you did in [Problem Set 2](#), except that your program this time should be written in Python.
 - Recall that the Coleman-Liau index is computed as $0.0588 * L - 0.296 * S - 15.8$, where L is the average number of letters per 100 words in the text, and S is the average number of sentences per 100 words in the text.
- Use `get_string` from the CS50 Library to get the user's input, and `print` to output your answer.
- Your program should count the number of letters, words, and sentences in the text. You may assume that a letter is any lowercase character from `a` to `z` or any uppercase character from `A` to `Z`, any sequence of characters separated by spaces should count as a word, and that any occurrence of a period, exclamation point, or question mark indicates the end of a sentence.
- Your program should print as output `"Grade X"` where X is the grade level computed by the Coleman-Liau formula, rounded to the nearest integer.
- If the resulting index number is 16 or higher (equivalent to or greater than a senior undergraduate reading level), your program should output `"Grade 16+"` instead of giving the exact index number. If the index number is less than 1, your program should output `"Before Grade 1"`.

Note that the specification here is only a summary of the requirements, so if you didn't do Readability in C, or if you are still unsure, we'd recommend that you review the [C specification and walkthrough](#) for clarification.

Usage

Your program should behave per the example below.

```
$ python readability.py
Text: Congratulations! Today is your day. You're off to Great Places! You're off and away!
Grade 3
```

Testing

No `check50` for this problem, but be sure to test your code for each of the following.

- Run your program as `python readability.py`, and wait for a prompt for input. Type in `One fish. Two fish. Red fish. Blue fish.` and press enter. Your program should output `Before Grade 1`.
- Run your program as `python readability.py`, and wait for a prompt for input. Type in `Would you like them here or there? I would not like them here or there. I would not like them anywhere.` and press enter. Your program should output `Grade 2`.

- Run your program as `python readability.py` , and wait for a prompt for input. Type in `Congratulations! Today is your day. You're off to Great Places! You're off and away!` and press enter. Your program should output `Grade 3` .
- Run your program as `python readability.py` , and wait for a prompt for input. Type in `Harry Potter was a highly unusual boy in many ways. For one thing, he hated the summer holidays more than any other time of year. For another, he really wanted to do his homework, but was forced to do it in secret, in the dead of the night. And he also happened to be a wizard.` and press enter. Your program should output `Grade 5` .
- Run your program as `python readability.py` , and wait for a prompt for input. Type in `In my younger and more vulnerable years my father gave me some advice that I've been turning over in my mind ever since.` and press enter. Your program should output `Grade 7` .
- Run your program as `python readability.py` , and wait for a prompt for input. Type in `Alice was beginning to get very tired of sitting by her sister on the bank, and of having nothing to do: once or twice she had peeped into the book her sister was reading, but it had no pictures or conversations in it, "and what is the use of a book," thought Alice "without pictures or conversation?"` and press enter. Your program should output `Grade 8` .
- Run your program as `python readability.py` , and wait for a prompt for input. Type in `When he was nearly thirteen, my brother Jem got his arm badly broken at the elbow. When it healed, and Jem's fears of never being able to play football were assuaged, he was seldom self-conscious about his injury. His left arm was somewhat shorter than his right; when he stood or walked, the back of his hand was at right angles to his body, his thumb parallel to his thigh.` and press enter. Your program should output `Grade 8` .
- Run your program as `python readability.py` , and wait for a prompt for input. Type in `There are more things in Heaven and Earth, Horatio, than are dreamt of in your philosophy.` and press enter. Your program should output `Grade 9` .
- Run your program as `python readability.py` , and wait for a prompt for input. Type in `It was a bright cold day in April, and the clocks were striking thirteen. Winston Smith, his chin nuzzled into his breast in an effort to escape the vile wind, slipped quickly through the glass doors of Victory Mansions, though not quickly enough to prevent a swirl of gritty dust from entering along with him.` and press enter. Your program should output `Grade 10` .
- Run your program as `python readability.py` , and wait for a prompt for input. Type in `A large class of computational problems involve the determination of properties of graphs, digraphs, integers, arrays of integers, finite families of finite sets, boolean formulas and elements of other countable domains.` and press enter. Your program should output `Grade 16+` .

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/2020/x/sentimental/readability
```


This is CS50x

OpenCourseWare

David J. Malan (<https://cs.harvard.edu/malan/>)

malan@harvard.edu

[f](https://www.facebook.com/dmalan/) (<https://www.facebook.com/dmalan/>) [G](https://github.com/dmalan/) (<https://github.com/dmalan/>) [@](https://www.instagram.com/davidjmalan/) (<https://www.instagram.com/davidjmalan/>) [in](https://www.linkedin.com/in/malan/)

(<https://www.linkedin.com/in/malan/>) [Q](https://www.quora.com/profile/David-J-Malan) (<https://www.quora.com/profile/David-J-Malan>) [T](https://twitter.com/davidjmalan/) (<https://twitter.com/davidjmalan/>)

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 \(https://ide.cs50.io/\)](https://ide.cs50.io/) and then, in a terminal window, execute each of the below.

- Execute `cd` to ensure that you're in `~/` (i.e., your home directory, aka `~`).
- If you haven't already, execute `mkdir pset6` to make (i.e., create) a directory called `pset6` in your home directory.
- Execute `cd pset6` to change into (i.e., open) that directory.
- Execute `wget https://cdn.cs50.net/2019/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: CTAGATAGATAGATAGATGACTA

Bob: CTAGATAGATAGATAGATAGATT

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.

Testing

No `check50` for this problem, but be sure to test your code 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`.

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/2020/x/dna
```