REGEX

* Regular expressions are mini-language for specifying text patterns. Writing code to do pattern matching without regular expressions is a huge pain.
* Regex strings often use backslashes (like \d), so they are often written using raw strings: r'\d'
* \d is the regex for a numeric digit character.
* Import the re module first.
* Call the re.compile() function to create a regex object.
* Call the regex object's search() method to create a match object.
* Call the match object's group() method to get the matched string.
* Groups are created in regex strings with parentheses.
* The first set of parentheses is group 1, the second is 2, and so on.
* Calling group() or group(0) returns the full matching string, group(1) returns group 1's matching string, and so on.
* Use \( and \) to match literal parentheses in the regex string.
* The | pipe can match one of many possible groups.
* The ? says the group matches zero or one times.
* The \* says the group matches zero or more times.
* The + says the group matches one or more times.
* The curly braces can match a specific number of times.
* The curly braces with two numbers matches a minimum and maximum number of times.
* Leaving out the first or second number in the curly braces says there is no minimum or maximum.
* "Greedy matching" matches the longest string possible, "nongreedy matching" (or "lazy matching") matches the shortest string possible.
* Putting a question mark after the curly braces makes it do a nongreedy/lazy match.
* The regex method findall() is passed a string, and returns all matches in it, not just the first match.
* If the regex has 0 or 1 group, findall() returns a list of strings.
* If the regex has 2 or more groups, findall() returns a list of tuples of strings.
* \d is a shorthand character class that matches digits. \w matches "word characters" (letters, numbers, and the underscore). \s matches whitespace characters (space, tab, newline).
* The uppercase shorthand character classes \D, \W, and \S match charaters that are not digits, word characters, and whitespace.
* You can make your own character classes with square brackets: [aeiou]
* A ^ caret makes it a negative character class, matching anything not in the brackets: [^aeiou]
* ^ means the string must start with pattern, $ means the string must end with the pattern. Both means the entire string must match the entire pattern.
* The . dot is a wildcard; it matches any character except newlines.
* Pass re.DOTALL as the second argument to re.compile() to make the . dot match newlines as well.
* Pass re.I as the second argument to re.compile() to make the matching case-insensitive.
* 