SUMMARY FOR REGULAR EXPRESSION

1. RE patterns compile into a series of bytecodes which are then executed by a matching engine written in C.
2. patterns and strings to be searched can be Unicode strings (str) as well as 8-bit strings (bytes) you cannot match a Unicode string with a byte pattern or vice-versa.
3. Here is the complete list of metacharacters: don't match themselves but change the meaning of RE. ^ $ \* + ? { } [ ] \ | ( ).
   1. '[' and ']' are used for specifying character class: a set of characters you wish to match. E.g. [abc] or [a-c] will match any of the characters a, b, or c. Metacharacters (except '\') are not active inside classes. E.g. [akm$] will match any of the characters a, k, m, $.
   2. '^' as the first character of the class is used for complementing the class. E.g. [^5] will match any character except 5 but [5^] will match any of the characters 5, ^. It is also used to match at the beginning of string. E.g. pattern ^From match in string “From Here to Eternity” but not in “Redirecting From Memory”
   3. “$”
   4. '\' most important: use as the first character.
      1. remove the special meaning of metacharacters. eg: \[ will match [ character in string.
      2. Some special sequences beginning with \ represent predefined sets of characters.
         1. \d matches any decimal digit [0-9]
         2. \D matches any non-digit character [^0-9]
         3. \s matches any whitespace character [\t\n\r\f\v]
         4. \S matches any non-whitespace character [^\t\n\r\f\v]
         5. \w matches any alphanumeric character [a-zA-Z0-9\_]
         6. \W matches any non alphanumeric character [^a-zA-Z0-9\_]
      3. . matches anything except a newline character. Alternate mode re.DOTALL where it will match any character
4. Repeating things: specify that portions of RE must be repeated a certain number of times.
   1. \* (greedy). E.g. ca\*t will match ct (0 matches), cat (1 match), caat (2 matches), and so on.
   2. + is the same as \* but + requires at least one occurrence. E.g. ca+t will match cat (1 match), caat (2 matches), and so on.
   3. ? will match either one or zero times. E.g. ca?t will match ct (0 matches) or cat (1 match).
   4. {m, n} m and n are int. there must be at least m and at most n repetitions eg: a/{1, 3}b will match a/b, a//b, and a///b. It won't match ab (no slashes) or a////b (4 slashes). Omitting m is interpreted as a lower limit of 0 while omitting n results in an upper bound of infinity. {0,} is the same as \*, {1,} is equivalent to +, and {0,1} is the same as ?. It’s better to use \*, +, or ?
5. Regular expressions are compiled into pattern objects using re.compile(RE). it also accepts optional flag arguments like re.compile(RE, re.IGNORECASE).
6. To match the exact '\section' you have to write \\section in RE.
7. RE will often be written in raw string notation. It helps to match a literal backslash and other metacharacters without difficulty.
8. Once you have a pattern object. you can call several methods on it like:
   1. match() Determine if the RE matches at the beginning of the string.
   2. search() Scan through a string, looking for any location where this RE matches.
   3. findall() Find all substrings where the RE matches, and return them as a list.
   4. finditer() Find all substrings where the RE matches, and return them as an iterator.
9. match() & search() function return match object for info about the match string. Each match from finditer() is also a match obj. match obj consists following functions:
   1. group() Return String matched by RE.
   2. start() Return starting position of the match.
   3. end() Return the ending position of the match.
   4. span() Return a tuple containing the (start, end) positions of the match
10. You don’t need to create a pattern object to call some module-level functions. E.g. match(), search(), findall(), sub() and etc.