KANAV BANSAL

REGULAR EXPRESSION:

A regular expression specifies a set of strings that matches it, the functions in this module let you check if a particular string matches a given regular expression (or if a given regular expression matches a particular string, which comes down to the same thing)

```
import re as regex
```

(or)

```
import re
```

APPLICATIONS OF REGULAR EXPRESSIONS:

1. Extraction of emails from a text document.
2. Regular Expressions for web scraping (Data collection).

3. Working with Date-Time features.

4. Using Regex for Text Pre-processing (NLP)

In simple,

↳ Validations of form in any online application.

↳ Pattern matching applications

Ex: ctrl+F → interpreter

↳ compilers, interpreters.

↳ TCP/IP, UDP

↳ Web scraping.

TCP/IP - TRANSMISSION CONTROL PROTOCOL / INTERNET PROTOCOL

A standard internet communications Protocol that allow digital computers to communicate over long distances.

UDP - USER DATA PROTOCOL:

A communications protocol that facilitates the exchange of messages between computing devices in a network. It's an alternative to the transmission control protocol (TCP).

In a network that uses the Internet Protocol (IP), it is sometimes referred to as UDP/IP.

METACHARACTERS:

↳ These are considered as the building blocks of regular expressions.

↳ These are the patterns used to match character combinations in the strings.

↳ It has a special meaning in finding patterns and are mostly used to define the search criteria and any text manipulations.

Metacharacters and its meaning:

[ ] → A set of characters

\ → Signals a special sequence
(can also be used to escape special characters)

. → Any character (except newline character)

^ → starts with

$ → Ends with

* → Zero or more occurances

+ → One or more occurances

{ } → Excatly the specified number of occurances.

| → Either or / start of alternative branch

( ) → capture & and group

? → extends the meaning of (, or 0/1 quantifier, or quantifier minimizer

CHARACTER CLASSES : (51)

A 'character class' or a 'character set', is a set of characters ~~part~~ kept in square brackets.

↳ the regex engine matches only one out of several characters in the character class or character set.

↳ We place the characters we want to match between square brackets.

USER-DEFINED CLASSES:

→ [0-9] - matches a single digit between 0 and 9.

→ [a-z] - matches a single alphabet between a and z which are in lower cases

→ [A-Z] - matches a single alphabet between A and Z which are in upper cases.

→ [a-zA-Z0-9] - matches alphanumerical characters

→ [^a-zA-Z0-9] - matches the special characters like ~, $, #, ! ..... (except alphanumerical)

→ [^abc] - match rest all alphabetical letters except a, b, c

PRE-DEFINED CHARACTER CLASSES:

\A - Returns a match if the specified characters are at the beginning of the string.

\b - Returns a match where the specified characters are at the beginning or at the end of the word.

\B - Returns a match where the specified characters are present, but NOT at the beginning (or at the end) of a word.

\d - Returns a match where the string contains digits (numbers from 0-9)

\D - Returns a match where the string DOES NOT contain digits.

\s - Returns a match where the string contains a white space character.

\S - Returns a match where the string DOES NOT contain a white space character.

\w - Returns a match where the string contains any word characters (from a-z, A-Z, digits from 0-9, and the underscore (_))

\W - Returns a match where the string DOES NOT contain any word characters, digits and underscore (^a-zA-Z0-9_)

\z - Returns a match if the specified characters are at the end of the string.

\t - Returns a match of tab character.

\n - Returns a match of a line-feed (new line) character.

QUANTIFIERS:

A Quantifier has the form {m,n} where m and n are the minimum and maximum times the expression to which the quantifier applies must match.

Example:

Both e{1,1} e{1,1} and e{2,2} match feel, but neither matches felt.

* → The component must be present either zero or more times.

+ → The component must be present either one or more times.

? → The component must be present either zero or one time.

{n} → The component must be present 'n' times.

{n,} → The component must be present atleast 'n' times.

{n,m} → The component must be present atleast 'n' times and no more than m times.

↳ Write a Regular Expression pattern for Python Identifier.

→ `[_a-zA-Z]\w+`