Regex (Regular Expression)

- Used to search & replace pattern, validate string follows a pattern
- Most underestimated tool. Found even in text editors
- Match done using:
  - Literal
  - Character class
  - Alteration
  - Metacharacters
  - Quantifier
  - Iteration
  - Groups
  - Look Arounds

Literals

- It's like normal text search
- Some literal holds special meaning. Example: ., *, +
- To match these special literal, we have to escape with \
- Example:
  - cat, dog, ram

Character Class

- Any character mentioned within big bracket is matched
- [AEIOUaeiou] gives match in letter if any character matches
- [A-Za-z] gives match for any alphabet
- ^ is used at initial to negate match.
- [^A-Za-z] matches everything except alphabet
- If we require to match ^, it has to be in middle/end [A-Z^]
- Example:
  - [crbh]at # Matches cat, rat, hat & bat. If I want a/o in 2nd?
  - [0-9] # Matches any digit, repeat 10 times for 10 digit

Alteration

- Used when we want to match multiple regex with or condition
- (r1|r2|r3) gives matches if either of r1, r2 or r3 matches

- Example:
  - (Mr |Mrs |Ms)[a-zA-z] # Matches designation & alphabet
  - (\+|)[0-9] # Matches digit with + at beginning
  - (Mr |Mrs |Ms)[a-zA-z]+ # Match designation & 1st name

Iteration

- Used if previous literal, class or group has to be matched n times
- Iteration is specified inside curly bracket {}.
- Single number denotes it has to repeat given number of times
- Number separated by , denotes min & max occurrence
- Example:
  - (cat){2} # Cat has to repeat 2 times for match
  - (cat){2, 4} # Cat can repeat 2 – 4 times for match
  - (cat){,4} # Cat can repeat 1-4 times for match
  - (cat){2,} # Cat can repeat 2-n times for match
  - [A-Z] [0-9]{9} # Alphabet has to be repeated by 9 digits for match
  - 9[0-9]{9} # 9 has to be followed by 9 digits for match

Special Character

\t	⇒	Tab
\n	⇒	New Line
\d	⇒	Any digit between 0 to 9
\D	⇒	Any non digit character
\s	⇒	White space. Space,Tab, New line
\S	⇒	Non white space
\w	⇒	Any alphanumeric character. 0-9,a-z,A-Z,_
\W	⇒	Non alphanumeric character
\b	⇒	Space around words
\B	⇒	Non word boundary. i.e no space around word
|	⇒	Matches regex expressions preceding or following it. Regex or
\	⇒	Escape character
.	⇒	Matches any character except newline
^	⇒	Match happens at beginning of string
[...]	⇒	Matches range of values inside the given bracket. Eg: [a-zA-Z\d]
(...)	⇒	Provide regex inside it and match for it
[^..]	⇒	Matches character that doesn't matches regex inside bracket
(R|S)	⇒	R and S are multiple regexes. Matches R or S
$	⇒	Match happens at the end of string

Quantifier

- Used to match multiple character in a string
- Greedy by default. Can use ? to make them non-greedy
- Some Quantifiers:
  - + ➔ One or more repetition of preceding regex
  - * ➔ Zero or more repetition of preceding regex
  - ? ➔ Zero or one repetition of preceding regex (question symbol)
- Example:
  - Mr[\.\s][A-Z][a-z]+ # Matches name with Mr. or Mr in its initial
  - M(r|s|rs)\.?\s[A-Z]\w* # Matches Mr, MS or Mrs

Group

- We can capture whole pattern in multiple parts with group
- We have two type of group: Capture & Non – Capture
- Capture group is in form (regex) & non-capture in (?:regex)
- Captured group are saved into memory & can reference later
- Example:
  - \d{4}([-\.\\/])?\d{2}\1\d{2} # Matches date
  - https?://(www\.)?(\w+)(\.\w+) # 2nd group match domain name
  - \b(\w+)\s+\1\b # Capture repeated word
  - (?:M(r|s|rs)\.?\s)([A-Z]\w*) # Non-Capturing Group

Look Ahead

- Look ahead is exclude in match but ensures it has to be there
- Looks forward in match
- Type: Positive & Negative
- Example:
  - regex_1(?=regex_2) ➔ regex_2 just after regex_1
  - regex_1(?!regex_2) ➔ regex_2 not after regex_1

Look Behind

- Look behind is exclude in match but ensures it has to be there
- Looks backward in match
- Type: Positive & Negative
- Example:
  - (?<=regex_1)regex_2 ➔ regex_1 just before regex_2
  - (?<!regex_1)regex_2 ➔ regex_1 not before regex_2

Examples

- (?<=Mr\.)\w+ ➔ Capture Name w/o title
- \w+(?=:\s) ➔ Capture key of key-value Pair
- (?<!hot)dog ➔ Capture dog if there is not hot prior
- \d+(?=kg) ➔ Captures the number if kg is afterwards
- (?<=@)\w+(?=\.(com|edu)) ➔ Capture the email provider

Anchor

- ^ & $ are anchors to bind the search
- ^ ➔ Match at beginning
- $ ➔ Match at end

Example:

- ^\d{10}$

Exercise

Python Implementation

- Python has built-in regex library
- Has functions:
  - search
  - findall
  - match
  - compile
  - split
  - sub
  - finditer

search

- Takes regex format and searches for it in a string
- Returns first substring that matches regex
- Invoke .group for getting matched string
- Returns None if not found

Example:

- import re
  usr_txt = “I bought 12kg apple”
  reg_to_match = r’\d+kg’
  matched_result = re.search(reg_to_match, usr_txt)
  print(matched_result.group())

Named Group

- For group, instead of using 1, 2 we can give name
- We can give name as ?P<group_name>

```
import re
usr_txt = "Date of today is 2025-07-10. I have 3 class today"
reg_to_match = r"(?P<Y>\d{4})-(?P<M>\d{2})-(?P<D>\d{2})"
matched_result = re.search(reg_to_match, usr_txt)
print(matched_result.group())
print(matched_result.group("Y"))
print(matched_result.group("M"))
print(matched_result.group("D"))
```

findall

- Search for pattern from start to end of string
- Returns all string that matches as a list

```
import re
usr_txt = "Date of today is 2025-07-10. I have 3 class today"
reg_to_match = r"\d{4}-\d{2}-\d{2}"
matched_result = re.findall(reg_to_match, usr_txt)
print(matched_result)
```

match

- Similar to search but match happens at the beginning

```
import re
usr_txt = "Date of today is 2025-07-10. I have 3 class today"
reg_to_match = r"\d{4}-\d{2}-\d{2}"
matched_result = re.match(reg_to_match, usr_txt)
print(matched_result.group())
```

compile

- We can compile regex pattern as object
- Compiled object can be used for future matches

Example:

- import re

  usr_txt = "I bought 12kg apple"

  reg_to_match = re.compile(r'\d+kg')

  matched_result = reg_to_match.search(usr_txt)

  print(matched_result.group())

split

- Splits string into list using regex as delimiter
- Compiled object can be used for future matches

Example:

- ```
  import re
  usr_txt = "I bought 12kg apple and 15kg orange"
  reg_to_match = re.compile(r'and')
  split_data = reg_to_match.split(usr_txt) # re.split(regex, word)
  print(split_data)
  ```
  ```

sub

- Substitute matched expression with given string

Example:

- import re

  usr_txt = "I bought 12kg apple and 15kg orange"

  reg_to_match = re.compile(r'and')

  replaced = re.sub(reg_to_match, '&', usr_txt)

  print(replaced)

finditer

- Find index and value in the match

Example

```
import re
text = 'Python exercises, PHP exercises, C# exercises'
pattern = '(\S)+?(?= exercises)'
for match in re.finditer(pattern, text):
    start_index = match.start()
    end_index = match.end()
    print("Found {} at index: {}-{}".format(match.group(), start_index, end_index))
```

Exercise