REGULAR EXPRESSIONS

TOPICS

1. DEFINITON
   - ⇨ Regular expressions are the collection/sequence of metacharacters, numbers, alphabets that defines a particular patterns like email, mobile no, etc
   - ⇨ These regular expressions are supported by the "re" module
2. META CHARACTERS, SPECIAL CHARACTERS, SETS

META CHARACTERS

Metacharacter	**Description**	**Example**
[]	Matches any one of the characters or range of characters inside the brackets.	[abc] or [a-z]
\	The backslash is used to escape special characters or to create special sequences.	"\r", "\s", "\w", "\d",
.	Matches any single character except newline.	"hel.o"
^	Matches the beginning of a string.	"^hello"
$	Matches the end of a string.	"welcome$"
*	Matches zero or more occurrences of the previous character.	"hello*"
+	Matches one or more occurrences of the previous character.	"hello+"
?	Matches zero or one occurrence of the previous character.	"he?llo"
{}	Curly braces are used to specify repetition.	"hello{2}"
|	Matches either the pattern before or after the | symbol	"hello|world"
()	Groups the enclosed pattern into a single unit.	

SPECIAL SEQUENCES

⇨ Special sequences are starts with \ symbol followed by one of the characters

Character	**Description**
\A	Matches the start of a string. It's similar to the ^ meta-character but differs in how it handles multiple lines.
\d	Matches any digit character [0-9].
\D	Matches any non-digit character.
\s	Matches any whitespace character (space, tab, newline, etc.).
\S	Matches any non-whitespace character.
\w	Matches any word character (letter, digit, or underscore).
\W	Matches any non-word character.
\Z	Matches the end of a string or the end of a line. It's similar to the $ meta-character, but differs in how it handles multiple lines.

SETS

- ⇨ Set is a group of character inside the square brackets that has special meaning

Character	**Description**
[abc]	Matches any one of the characters a, b, or c.
[^abc]	Matches any character that is not a, b, or c.
[a-z]	Matches any lowercase letter from a to z.
[A-Z]	Matches any uppercase letter from A to Z.
[0123]	Matches if the string contains any of the specified digits.
[0-9]	Matches any digit character from 0 to 9.
[0-5][0-9]	Matches if the string contains any digit between 00 and 59.
[a-zA-Z]	Matches if the string contains any alphabet (lower-case or upper-case).
[a-zA-Z0-9]	Matches any alphanumeric character (letter or digit).

3. REGEX FUNCTIONS

⇨ The “re” module provides the methods to perform operations on the regular expressions

1. search()
   a. this method is used to search for a particular pattern in a given input/string
   b. if more than one matches are occurred, then first occurrence of match is returned
   c. if there is no match, None is returned

   syntax:

   ```
   re.search(pattern,string)
   ```

   ex:

   ```
   import re
   pattern=r"is"
   str="python is very easy"
   matchobj=re.search(pattern,str)
   if matchobj:
       print("match is found")
       print(f"pattern {pattern} start at
   {matchobj.start()} and end's at
   {matchobj.end()} and matched pattern=
   {matchobj.group()}")
   else:
       print("match is not found")
   ```

⇨ MATCH OBJECT METHODS

- ➢ start() – it returns the starting index of the matched pattern
- ➢ end() – it returns the end index+1 of the matched pattern
- ➢ group() – it returns the matched pattern

2. match()
   a. this method is used to check whether the pattern is matching at the beginning of the given input/string
   b. if the match is found, it returns the matched pattern
   c. if the match is not found, it return None

   syntax:

```
re.match(pattern,string)
```

   ex:

```
import re
pattern=r'hello world'
str="hello"
matchobj=re.match(pattern,str)
if matchobj:
    print(f" matched
pattern={matchobj.group()}")
else:
    print("match is not found")
```

3. findall()
   a. this method is used to find all the occurences of a pattern in a given input/string
   b. it returns all the matched patterns as a list format
   c. if no match is found, it returns the empty list

syntax:

```
re.findall(pattern,string)
```

ex:

```
import re

pattern=r"cow"
str="that is cow this is cow here is the cow
there is the cow"
match_list=re.findall(pattern,str)
if match_list:
      print(match_list)
else:
   print("match is not found")
```

4. split()
   a. this method is used to split the given input/string into list of sub strings based on the specified pattern
      syntax:

```
re.split(pattern,string)
```

      ex:

```
import re
pattern=r"\s+"
str="python java c c++"
sub_strings=re.split(pattern,str)
if sub_strings:
    print(sub_strings)
else:
    print("not fouund")
```

5. sub()
   a. this method is used to substitute the new string in place of matched pattern in a given input/string
      syntax:

```
re.sub(pattern,replacement_string,string)
```

      ex:

```
import re
pattern=r'\s+'
replacement_str="="
str="this is python"
new_str=re.sub(pattern,replacement_str,str)
if new_str:
      print(new_str)
else:
   print("not found")
```

4. VALIDATING DATA USING REGULAR EXPRESSIONS
   ⇨ We can validate the data by using regular expressions

Ex:

Validating integer

```
import re
num=input("enter the number")
pattern=r'^[+-]?\d+$'
if re.match(pattern,num):
    print("valid integer")
else:
    print("invalid integer")
```