Regular expressions

Regular expressions (regex) are a powerful tool used for pattern matching and manipulating strings. In Python, you can work with regular expressions using the `re` module, which provides functions and methods for working with regex.

To **use regular expressions** in Python, you need to **import the re module**:

```
import re
```

Commonly used Functions in re module

Here are some commonly used functions and methods provided by the **re module**:

1. **re.search(pattern, string)**: This function searches the given string for a match to the specified pattern and returns a match object if there is a match. For example:

```
import re

string = "Hello, world!"
pattern = r"world"
match = re.search(pattern, string)
if match:
    print("Match found!")
else:
    print("Match not found!")
```

Output:

Match found!

2. **re.match(pattern, string):** This function checks if the pattern matches at the beginning of the string. It returns a match object if there is a match; otherwise, it returns None. For example:

```
import re

string = "Hello, world!"
pattern = r"Hello"
match = re.match(pattern, string)
if match:
    print("Match found!")
else:
    print("Match not found!")
```

Output:

Match found!

3. **re.findall(pattern, string):** This function finds all occurrences of the pattern in the string and returns them as a list of strings. For example:

```
import re

string = "Hello, hello, hello!"
pattern = r"hello"
matches = re.findall(pattern, string)
print(matches)
Output:
['hello', 'hello', 'hello']
```

4. **re.sub(pattern, replacement, string):** This function replaces occurrences of the pattern in the string with the specified replacement string and returns the modified string. For example:

```
import re

string = "Hello, world!"
pattern = r"world"
replacement = "Python"
new_string = re.sub(pattern, replacement, string)
print(new_string)
Output:
Hello, Python!
```

5. **re.compile()** function to create a pattern object and then apply it to the string using various methods like search(), match(), or findall(). Here's an example:

```
import re
pattern=re.compile('TP')
result=pattern.findall('TP VIT TP')
print (result)
Output:
['TP', 'TP']
```

6. **re.split(pattern, string, maxsplit)** function split the string by occurrences of the regex pattern and return the list containing the resulting substrings.

For an example1:

```
import re
string='one-1 two-2 twenty-20'
Pattern = '\D+'
result=re.split(Pattern,string)
print result
Output:
    ['', '1', '2', '20']
```

Example2

```
import re
string='one-1 two-2 twenty-20'
Pattern = '\D+'
result=re.split(Pattern,string,maxsplit=2) // maxsplit =2 maximum number of 2 splits will occur
print (result)

 Output:
['', '1', '2 twenty-20']
```

Pattern Syntax

Regular expressions use a specific syntax to define patterns. Here are some common elements and syntax rules used in regular expressions:

1. **Literal Characters:**
    - Most characters in a regular expression match themselves literally. For example, the pattern `a` matches the character "a" in the input text.
2. **Metacharacters:**
    - Metacharacters have special meanings in regular expressions. Some common metacharacters include:
        - `.` **(dot):** Matches any character except a newline.
        - `^` **(caret):** Matches the start of a string.
        - `$` **(dollar):** Matches the end of a string.
        - `*` **(asterisk):** Matches zero or more occurrences of the preceding character or group.

- **`+` (plus):** Matches one or more occurrences of the preceding character or group.
- **`?` (question mark):** Matches zero or one occurrence of the preceding character or group.
- **`[]` (square brackets):** Defines a character set.
- **`()` (parentheses):** Groups patterns together.

3. **Character Classes:**
   - Character classes define sets of characters to match. Some common examples include:
     - **`[abc]`:** Matches any single character "a", "b", or "c".
     - **`[0-9]`:** Matches any digit from 0 to 9.
     - **`[^a-z]`:** Matches any character that is not a lowercase letter.
     - [012]: Matches any single number "0", "1", or "2".
     - [a-z]: Matches any lower case character between a to z
     - [A-Za-z]: Matches any lower case or upper case character between a to z
     - [+]: Matches any character + in the string
     - [0-9][0-9]: Matches any two digit numbers from 00 and 99
4. **Quantifiers:**
   - Quantifiers specify the number of occurrences of a character or group. Some common quantifiers include:
     - **`{n}`:** Matches exactly n occurrences.
     - **`{n,}`:** Matches n or more occurrences.
     - **`{n,m}`:** Matches between n and m occurrences.
5. **Escape Sequences:**
   - Some characters have special meanings in regular expressions. To match them literally, you need to use escape sequences. The escape character is the backslash `\`. For example, to match a literal dot, you would use `\.`.

Summary of Pattern Syntax

Syntax	Description	Example
`.`	Matches any character except a newline	`a.b` matches `axb`, `a0b`
`^`	Matches the start of a string	`^Hello` matches `Hello, World!` at the start
`$`	Matches the end of a string	`World!$` matches `Hello, World!` at the end

Syntax	Description	Example
*	Matches zero or more occurrences of the preceding character	ab*c matches ac, abc, abbc
+	Matches one or more occurrences of the preceding character	ab+c matches abc, abbc, but not ac
?	Matches zero or one occurrence of the preceding character	colou?r matches color and colour
[]	Defines a character set	[aeiou] matches any vowel character
[^]	Defines a negated character set	[^0-9] matches any non-digit character
()	Groups patterns together	(abc)+ matches abc, abcabc, abcabcabc

Note: Remember to use the re module in Python and its functions like match(), search(), or findall() to apply these regular expressions to actual text strings.

Special Sequences for pattern Syntax

A special sequence is a \ followed by one of the characters in the list below, and has a special meaning:

Special Sequence	Description	Example
\d	Matches any digit (0-9)	\d+ matches 123, 4567, etc.
\D	Matches any non-digit character	\D+ matches abc, xyz, etc.
\w	Matches any alphanumeric character and underscore	\w+ matches hello, world_123
\W	Matches any non-alphanumeric character	\W+ matches !@#$%, .,;:
\s	Matches any whitespace character (space, tab, newline)	\s+ matches " ", \t, \n
\S	Matches any non-whitespace character	\S+ matches hello, world!
\b	Matches a word boundary (position between \w and \W)	\b\w+\b matches whole words

Special Sequence	Description	Example
\B	Matches a non-word boundary	\B\w\B matches word characters within a word

These special sequences can be used within a regular expression pattern to match specific types of characters or positions in a string.

For example, the pattern \d+ will match one or more digits, while \w+ will match one or more alphanumeric characters and underscores.

You can combine these special sequences with other regular expression syntax to create more powerful and specific patterns for matching and manipulating text.

Example 1 : Matching Words Starting with a Specific Letter:

```
import re

pattern = r"\bA\w+\b"

text = "Alice is a great artist. Andy loves apples."

matches = re.findall(pattern, text)

print(matches)
```

Output: ['Alice', 'artist', 'Andy', 'apples']

Example 2 : Counting the Occurrences of a Specific Character:

```
import re

pattern = r"a"

text = "The cat sat on the mat."

occurrences = len(re.findall(pattern, text))

print(occurrences)
```

Output: 2

Example 3 : Extracting Numbers from a String:

```
import re

pattern = r"\d+"

text = "I have 10 apples and 5 oranges."

numbers = re.findall(pattern, text)

print(numbers)
```

Output: ['10', '5']

Example 4: Splitting a String by Punctuation:

```
import re

pattern = r"[.,;!?]"

text = "Hello, how are you? I am fine."

segments = re.split(pattern, text)

print(segments)
```

Output: ['Hello', ' how are you', ' I am fine', '']

Example 5: Validate an Email Address:

```
import re

def validate_email(email):
    pattern = r'^[\w\.-]+@[\w\.-]+\.\w+$'
    match = re.match(pattern, email)
    if match:
        return True
    else:
        return False

# Example usage:
```

email1 = 'example@example.com'

email2 = 'invalid.email@com'

print(validate_email(email1)) # Output: True

print(validate_email(email2)) # Output: False

In the above function, we use the regular expression pattern `r'^[\w\.-]+@[\w\.-]+\.\w+$'` to match the email address. Let's break down the pattern:

- `^` asserts the start of the string.
- `[\w\.-]+` matches one or more word characters (letters, digits, or underscores), dots, or hyphens. This represents the username part of the email address.
- `@` matches the literal "@" symbol.
- `[\w\.-]+` matches one or more word characters, dots, or hyphens. This represents the domain name part of the email address.
- `\.` matches the literal dot symbol.
- `\w+` matches one or more word characters. This represents the top-level domain (e.g., com, org, edu) part of the email address.
- `$` asserts the end of the string.

Example 6: Extract All Phone Numbers from a String:

```
import re

def extract_phone_numbers(text):
    pattern = r'\d{3}-\d{3}-\d{4}'
    matches = re.findall(pattern, text)
    return matches

# Example usage:
text = 'Contact us at 123-456-7890 or 987-654-3210'
phone_numbers = extract_phone_numbers(text)
print(phone_numbers) # Output: ['123-456-7890', '987-654-3210']
```

Example 7: Matching Email Addresses:

```
import re
```

pattern = r"\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}\b"

text = "Contact us at info@example.com or support@website.co"
matches = re.findall(pattern, text)

print(matches)
Output: ['info@example.com', 'support@website.co']
Example 8: Extracting Dates:
import re

pattern = r"\d{2}/\d{2}/\d{4}"

text = "Meeting scheduled for 05/23/2023. Please confirm your availability."
matches = re.findall(pattern, text)

print(matches)
Output: ['05/23/2023']
Example 9: Splitting Text into Sentences:
import re

pattern = r"(?<!\w\.\w.)(?<![A-Z][a-z]\.)(?<=\.|\?)\s"

text = "Hello! How are you? I'm doing fine. What about you?"
sentences = re.split(pattern, text)

print(sentences)
Output: ['Hello!', 'How are you?', "I'm doing fine.", 'What about you?']
Example 10: Extracting URLs:
import re

pattern = r"http[s]?://(?:[a-zA-Z]|[0-9]|[$-_@.&+]|[!*\\(\\),]|(?:%[0-9a-fA-F][0-9a-fA-F]))+"

text = "Check out our website at https://example.com for more information."

matches = re.findall(pattern, text)

print(matches)
Output: ['https://example.com']

Example 11:
Validate Gender using Regular Expressions
Correct format
Male / male / MALE / M / m
Female / female / FEMALE / F / f
Not prefer to say

```
import re
str = "m"
# Regex to check valid GENDER
regex = "(m|M|male|Male|f|F|female|Female|FEMALE|MALE|Not prefer to say)$"

# Compile the ReGex
p = re.compile(regex)
if(re.search(p, str)):
        print("True")
else:
        print("False")
```

Output: True

Example 12:
To find the sequences of one upper case letter followed by lower case letters.

```
import re
str = "Welcome"

regex = "[A-Z]+[a-z]+$$"

if(re.search(regex, str)):
```

```
    print("True")
else:
    print("False")
```

Output

True

Exercise Problem

1. Validating Time in HH:MM Format: Write a Python program that prompts the user to enter a time in the format HH:MM and validates whether it is a valid time format using regular expressions. Display a message indicating whether the time is valid or not.
2. Extract Domain Names: Write a Python function `extract_domains(text)` that takes a string `text` as input and returns a list of unique domain names found in the text. Consider that domain names can start with either `http://` or `https://`, and they should be followed by alphanumeric characters, dots, or hyphens. For example, given the text "Check out my website at http://www.example.com or visit https://www.openai.com for more information," the function should return `['www.example.com', 'www.openai.com']`.
3. Validate Phone Numbers: Write a Python function `validate_phone_numbers(numbers)` that takes a list of strings `numbers` as input and returns a list of valid phone numbers. A valid phone number should consist of three groups of digits separated by hyphens, where each group contains exactly three digits. For example, given the list `['123-456-7890', '987-654-321', '456-7890', '123-45-678']`, the function should return `['123-456-7890']`.
4. Extract Hashtags: Write a Python function `extract_hashtags(text)` that takes a string `text` as input and returns a list of unique hashtags found in the text. A hashtag is defined as a word starting with the '#' symbol, followed by alphanumeric characters. For example, given the text "I love #Python and #programming in general. #CodeIsFun!", the function should return `['#Python', '#programming', '#CodeIsFun']`.
5. Extract Email Domains: Write a Python function `extract_email_domains(emails)` that takes a list of strings `emails` as input and returns a list of unique email domains. An email domain is the part of the email address that follows the '@' symbol. For example, given the list `['example1@example.com', 'example2@openai.com', 'example3@gmail.com']`, the function should return `['example.com', 'openai.com', 'gmail.com']`.
6. Validating ZIP Codes: Write a Python program that prompts the user for a ZIP code and validates whether it is a valid ZIP code format using regular expressions. Display a message indicating whether the ZIP code is valid or not.
7. Extracting Words Starting with a Specific Letter: Write a Python function that takes a string and a letter as input and extracts all the words from the string that start with the given letter using regular expressions. Return a list of the extracted words.

8. Validating Password Complexity: Write a Python function that takes a password as input and validates whether it meets the following complexity criteria using regular expressions:

- At least 8 characters long
- Contains at least one uppercase letter
- Contains at least one lowercase letter
- Contains at least one digit

Return `True` if the password meets the criteria, and `False` otherwise.

9. Counting Vowels: Write a Python function that takes a string as input and counts the number of vowels (a, e, i, o, u) in the string using the `re` module. Return the count.
10. Splitting Text by Non-Alphanumeric Characters: Write a Python function that takes a string as input and splits it into a list of words by using non-alphanumeric characters as separators. Use the `re` module to split the text.
11. Counting Words: Write a Python function that takes a string as input and counts the number of words in the string using the `re` module. Consider a word to be any sequence of non-whitespace characters. Return the count.
12. Extracting Numbers from a String: Write a Python function that takes a string as input and extracts all the numbers (integer or decimal) from it using the `re` module. Return a list of the extracted numbers.