In Python, the re module provides support for working with **regular expressions** (regex). Regular expressions are powerful tools for matching patterns within strings. You can use them for searching, splitting, and replacing parts of text.

**Basic Functions in re**

1. **re.search()**:
   * **Definition**: Searches a string for a match and returns the first match object if found; otherwise, returns None.
   * **Syntax**:

re.search(pattern, string, flags=0)

* + **Example**:

Import re

text = "I love Python programming!"

match = re.search(r'Python', text)

if match:

print ("Pattern found:", match.group())

**Output**:

Pattern found: Python

**Example 2:**

import re

a = input("Enter The pattern : ")

b = re.search(a,"abcadefgfg/")

if b!= None:

    print("Match found for search ")

    print("StartIndex:", b.start(),"End Index:", b.end())

else:

    print("No Match found")

**2.re.match()**:

* + **Definition**: Checks for a match only at the **beginning** of the string. If it matches, returns a match object; otherwise, returns None.
  + **Syntax**:

re.match(pattern, string, flags=0)

* + **Example**:

import re

text = "Python is great!"

match = re.match(r'Python', text)

if match:

print("Match at the start:", match.group())

else:

print("No match at the start")

**Output**:

sql

Copy code

Match at the start: Python

**Example 2:**

import re

a = input("Enter The pattern : ")

b = re.match(a,"abcadefd")

if b!= None:

    print("Match found")

    print("StartIndex:", b.start(),"End Index:", b.end())

else:

    print("No Match found")

**3.re.findall()**:

* + **Definition**: Returns in form of List if the data exits.
  + **Syntax**:

re.findall(pattern, string, flags=0)

* + **Example**:

import re

text = "Python is fun, and Python is powerful."

matches = re.findall(r'Python', text)

print("All matches:", matches)

**Output**:

All matches: ['Python', 'Python']

**Example 2:**

import re

l = re.findall("[0-9]", "az7k F9q$.")

print(l)

**4.re.finditer()**:

* + **Definition**: Returns an iterator yielding match objects for all non-overlapping matches in the string.
  + **Syntax**:

re.finditer(pattern, string, flags=0)

* + **Example**:

python

Copy code

import re

text = "Python is fun, and Python is powerful."

matches = re.finditer(r'Python', text)

for match in matches:

print("Match found at position:", match.start())

**Output**:

arduino

Copy code

Match found at position: 0

Match found at position: 18

**5.re.split()**:

* + **Definition**: Splits the string by occurrences of the pattern and returns a list of substrings.
  + **Syntax**:

re.split(pattern, string, maxsplit=0, flags=0)

* + **Example**:

import re

text = "apple, banana, cherry, date"

result = re.split(r', ', text)

print("Split result:", result)

**Output**:

Split result: ['apple', 'banana', 'cherry', 'date']

import re

a = re.split("," , "apple,banana,cherry")

print(a)

6.**re.sub()**:

* + **Definition**: Substitutes all occurrences of the pattern in the string with the replacement string.
  + **Syntax**:

re.sub(pattern, repl, string, count=0, flags=0)

* + **Example**:

python

Copy code

import re

text = "Cats are cute. Cats are playful."

result = re.sub(r'Cats', 'Dogs', text)

print("Substitution result:", result)

**Output**:

sql

Copy code

Substitution result: Dogs are cute. Dogs are playful.

**Example 2:**

import re

a = re.sub("[a-z]", "&", "Hi@m3")

print(a)

**7.** **subn():**

Definition: The re.subn() function in Python is similar to re.sub(), but with an added benefit: it returns both the new string and the number of replacements made.

This function can be useful when you want to know not just the result of the substitutions but also how many times the pattern was replaced in the string.

re.sub(pattern, repl, string, count=0, flags=0)

it result in tuple

0-represnts the reslut string

1-represnts the count of replacement

**Example:**

import re

text = "I have 2 apples, 3 bananas, and 5 oranges."

# Replace digits with the word 'fruit'

result = re.subn(r'\d+', 'fruit', text)

print("Resulting string:", result[0])

print("Number of substitutions:", result[1])

**Output:**

**Resulting string: I have fruit apples, fruit bananas, and fruit oranges.**

**Number of substitutions: 3**

**Example 2:**

s = re.subn("[a-z]","#","aBc7@w")

print(s)

print("The result of a string",s[0])

print("The no of occurences",s[1])

### 8. re.fullmatch()

* **Definition**: re.fullmatch() checks whether the **entire** string matches a pattern. If the full string matches the pattern, it returns a match object; otherwise, it returns None.
* **Syntax**:

python

Copy code

re.fullmatch(pattern, string, flags=0)

* + **pattern**: The regex pattern to match.
  + **string**: The input string to be checked.
  + **flags**: Optional flags to modify the matching behavior.
* **Use Case**: If you want to ensure that the whole string matches a regex pattern, not just a part of it.
* **Example**:

python

Copy code

import re

pattern = r'\d{4}-\d{2}-\d{2}' # Pattern for a date in the format YYYY-MM-DD

date = "2023-09-24"

match = re.fullmatch(pattern, date)

if match:

print("Full match!")

else:

print("Not a full match.")

**Output**:

sql

Copy code

Full match!

**Explanation**: The string "2023-09-24" fully matches the pattern for a date in the format YYYY-MM-DD.

* **Partial Match** Example:

date = "2023-09-24 and some text"

match = re.fullmatch(pattern, date)

if match:

print("Full match!")

else:

print("Not a full match.")

**Output**:

css

Copy code

Not a full match.

Example 2:

import re

a = input("Enter The pattern : ")

b = re.fullmatch(a,"abcadefgd")

if b!= None:

    print("Match found ")

    print("StartIndex:", b.start(),"End Index:", b.end())

else:

    print("No Match found")

**9.** escape()**:**

**Definition**: re.escape() escapes all non-alphanumeric characters in a string so that they can be used in a regex pattern **literally**. This is helpful when you need to match a string that contains special characters like . or \*, which otherwise have special meaning in regular expressions.

 **Syntax**:

python

Copy code

re.escape(string)

 **Use Case**: When you want to treat special characters in a string literally, without them being interpreted as regex operators.

 **Example**:

python

Copy code

import re

text = "Special characters: \* and ?."

escaped\_text = re.escape(text)

print("Escaped string:", escaped\_text)

**Output**:

c

Copy code

Escaped string: Special\ characters:\ \\*\ and\ \?\.

**Explanation**: The characters \*, ?, and . are special in regex. By using re.escape(), we can turn them into literals, ensuring that they are treated as ordinary characters, not regex operators.

 **Use in Regex**:

python

Copy code

pattern = re.escape("file.name?")

match = re.search(pattern, "Check the file file.name? in the directory")

if match:

print("Pattern found:", match.group())

**Output**:

mathematica

Copy code

Pattern found: file.name?

**Explanation**: The function escapes the special characters, ensuring that file.name? is matched literally rather than using the ? as a regex operator.

**Commonly Used Flags**

* **re.IGNORECASE**: Makes the pattern case-insensitive.
* **re.MULTILINE**: Allows ^ and $ to match the start and end of each line in the string.
* **re.DOTALL**: Allows . to match any character, including a newline.

**Example with Flags**

python

Copy code

import re

text = "Hello WORLD, hello world!"

result = re.findall(r'hello', text, re.IGNORECASE)

print("Case-insensitive matches:", result)

**Output**:

sql

Copy code

Case-insensitive matches: ['Hello', 'hello']

**Summary**

| **Function** | **Use Case** | **Returns** |
| --- | --- | --- |
| re.search() | Search for a pattern in a string | Match object (first match) |
| re.match() | Match pattern at the start of string | Match object (if match at start) |
| re.findall() | Find all matches of a pattern | List of matched strings |
| re.finditer() | Find all matches as match objects | Iterator of match objects |
| re.split() | Split string by pattern | List of substrings |
| re.sub() | Replace pattern with a new string | New string with replacements |

This covers the most common functions of the re module and how they are used to work with regular expressions in Python.

4o