کتابخانه عبارت منظم مورد استفاده ما در این پروژه regex در پایتون است.

عبارات با قاعده (Regular expression) یا به اختصار RegEx، رشته‌ای حاوی کاراکترهایی خاص و با معنی است و در واقع با استفاده از RegEx می‌توان یک الگو (pattern) برای جستجو در متن یا تطابق آن ایجاد کرد.

ماژول re یک مجموعه ابزار برای کار با RegEx در پایتون را فراهم آورده است که می‌توان آن را موتور تحلیل RegEx در پایتون دانست.

این پروژه از کتابخانه Python re (عبارت منظم) برای تعریف مجموعه ای از الگوهای عبارت منظم استفاده می کند و سپس این الگوها را در لیستی از رشته های ورودی اعمال می کند تا مطابقت را بررسی کند.

**الگوریتم توکن سازی( Tokenization) در regex**

توکن سازی (Tokenization) فرآیند شکستن یک متن به واحدهای مجزا، مانند کلمات یا عبارات است. الگوریتم مورد استفاده برای توکن سازی شامل مراحل زیر است:

* پیش پردازش متن: متن ورودی تحت پیش پردازش اولیه، از جمله حروف کوچک و مدیریت کاراکترهای خاص قرار می گیرد.
* توکن‌سازی مبتنی بر قانون : regex از توکن‌سازی مبتنی بر قانون استفاده می‌کند، که در آن قوانین از پیش تعریف‌شده نحوه تقسیم متن به توکن‌ها را تعیین می‌کنند. این قوانین علائم نگارشی، فاصله ها و قراردادهای خاص زبان را در نظر می گیرند.
* توکن سازی فضای خالی: فضای خالی باقیمانده برای جداسازی توکن ها استفاده می شود. این مرحله تضمین می کند که خروجی فاصله اصلی در متن را حفظ می کند.

**بررسی پذیرش / رد رشته ها با استفاده از عبارات منظم:**

عبارات منظم (regex) ابزار قدرتمندی برای تطبیق الگو در رشته ها هستند. برای بررسی پذیرش یا رد رشته ها بر اساس مجموعه ای از عبارات منظم، می توان از الگوریتم زیر استفاده کرد:

< مرحله اول > *کامپایل عبارات منظم:* لیستی از عبارات منظم ایجاد میکنیم و با استفاده از تابع re.compile() در پایتون آنها را کامپایل کنید. کامپایل عملکرد را بهبود می بخشد، به خصوص زمانی که به طور مکرر الگوهای یکسانی را به رشته های متعدد اعمال می کنیم.

regex\_patterns = [  
 re.compile(r'^((ab)\*(cd)\*)\*$'),  
 re.compile(r'^(ab+cd)\*$'),  
 re.compile(r'^(a+ba)\*(b+la)$'),

< مرحله دوم >  *تعریف رشته های ورودی :* لیستی از رشته های ورودی ایجاد میکنیم تا بتوانیم روی هرکدام از انها پردازش انجام دهیم.

regex\_patterns = [  
 re.compile(r'^((ab)\*(cd)\*)\*$'),  
 re.compile(r'^(ab+cd)\*$'),  
 re.compile(r'^(a+ba)\*(b+la)$'),

< مرحله سوم > *پردازش ورودی:* برای هر رشته ورودی، از تابع re.match() یا re.fullmatch() استفاده میکنیم تا بررسی کنیم که آیا با یکی ازعبارات منظم کامپایل شده در لیست ما مطابقت دارد یا خیر. این توابع اگر الگویی در ابتدا (for match()) یا در کل رشته (for fullmatch()) یافت شود، یک شی مطابق را برمی گرداند.

def match\_strings\_with\_regex(input\_strings, regex\_patterns):  
 for i, string in enumerate(input\_strings, start=1):  
 matches = []  
 for j, pattern in enumerate(regex\_patterns, start=1):  
 if pattern.match(string):  
 matches.append(f"Expression {j}")  
 print(f"String {i}: '{string}' matches with {', '.join(matches) if matches else 'none'}")

match\_strings\_with\_regex(input\_strings, regex\_patterns)

< مرحله چهارم > *تفسیر خروجی:* بر اساس نتایج تطبیق عبارت منظم، تعیین میکنیم که کدام الگوها، در صورت وجود، با هر رشته ورودی مطابقت دارند. این اطلاعات به درک اینکه کدام عبارات منظم رشته های خاص را می پذیرند یا رد می کنند کمک می کند.

Result:

String 1: 'abcdababcdcdab' matches with Expression 1

String 2: 'acdcdcdb' matches with none

String 3: 'aaaaaaabaaaaa' matches with none