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 > *تعریف 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.

expression\_match(text: str):

patterns = [("pattern\_1",r"((ab)\*(cd)\*)\*"),

("pattern\_2",r"(ab+cd)\*"),

("pattern\_3",r"(aa)\*b(aa)\*+a(aa)\*ba(aa)\*"),

("pattern\_4",r"(a+ba)\*(b+la)"),

("pattern\_5",r"(a\*baa\*)\*(b+la)+a\*(b+la)"),

("pattern\_6",r"(aa)\*(ba+ab)(bb+(ba+ab)(aa)\*(ba+ab))\*")]

< مرحله سوم > *پردازش ورودی:* برای هر رشته ورودی، بررسی میکنیم که آیا با یکی ازعبارات منظم کامپایل شده در لیست ما مطابقت دارد یا خیر. این توابع اگر الگویی در ابتدا یا در کل رشته بیابند، یک شی مطابق را برمی گرداند.

matches = []  
for pattern\_name, pattern in patterns:  
 if re.match(pattern, text):  
 matches.append(pattern\_name)  
  
if len(matches) == 0:  
 print("the didn't match to any expression")  
else:  
 print(f"matched expressions:{matches}")

< 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.

def main():  
 text = input()  
 expression\_match(text)  
   
if \_\_name\_\_ =='\_\_main\_\_':  
 main()

Result:

String 1: 'abcdababcdcdab' matches with Expression 1

String 2: 'acdcdcdb' matches with none

String 3: 'aaaaaaabaaaaa' matches with none

String 4: 'ababababab' matches with Expression 1

String 5: 'aabbaaaabaaab' matches with none

String 6: 'ababababababbbbbb' matches with none

:3

Pattern\_1 , pattern\_2 یکسان هستند

Pattern\_3 , pattern\_4 یکسان هستند

عبارات منظم با ساختار ساده‌تر و کمترین تعداد انتخاب‌ها (alternatives) و بازگرداندهای تکراری معمولاً سریعتر هستند. به عنوان مثال، عبارت منظم "(ab|cd)\*" در pattern\_2 به نظر می‌رسد که نسبت به سایر الگوها سریعتر باشد، زیرا تنها از دو انتخاب ad یا cd استفاده می‌کند و تکرارهای متناوب را بررسی می‌کند.

abcdababcdcdab

matched expressions:['pattern\_1', 'pattern\_2']

Time taken for pattern\_1: 0.071508 seconds

Time taken for pattern\_2: 0.050320 seconds

Time taken for pattern\_4: 0.022569 seconds

Time taken for pattern\_5: 0.023762 seconds

Time taken for pattern\_6: 0.023202 seconds

acdcdcdb

the didn't match to any expression

Time taken for pattern\_1: 0.023419 seconds

Time taken for pattern\_2: 0.017573 seconds

Time taken for pattern\_3: 0.020060 seconds

Time taken for pattern\_4: 0.021050 seconds

Time taken for pattern\_5: 0.021276 seconds

Time taken for pattern\_6: 0.017185 seconds

aaaaaaabaaaaa

matched expressions:['pattern\_3']

Time taken for pattern\_1: 0.023800 seconds

Time taken for pattern\_2: 0.017806 seconds

Time taken for pattern\_4: 0.073125 seconds

Time taken for pattern\_5: 0.051060 seconds

Time taken for pattern\_6: 0.033055 seconds

ababababab

matched expressions:['pattern\_1', 'pattern\_2', 'pattern\_4', 'pattern\_5', 'pattern\_6']

Time taken for pattern\_1: 0.053860 seconds

Time taken for pattern\_2: 0.031387 seconds

Time taken for pattern\_4: 0.039764 seconds

Time taken for pattern\_5: 0.051984 seconds

Time taken for pattern\_6: 0.056546 seconds

aabbaaaabaaab

the text didn't match to any expression

Time taken for pattern\_1: 0.025325 seconds

Time taken for pattern\_2: 0.017129 seconds

Time taken for pattern\_4: 0.025474 seconds

Time taken for pattern\_5: 0.026224 seconds

Time taken for pattern\_6: 0.021486 seconds

ababababababbbbbb

the text didn't match to any expression

Time taken for pattern\_1: 0.853707 seconds

Time taken for pattern\_2: 0.048609 seconds

Time taken for pattern\_3: 0.022301 seconds

Time taken for pattern\_4: 0.060012 seconds

Time taken for pattern\_5: 0.068140 seconds

Time taken for pattern\_6: 0.063606 seconds