Expressões Regulares com Python

Fundamentos Teóricos da Computação (FTC)

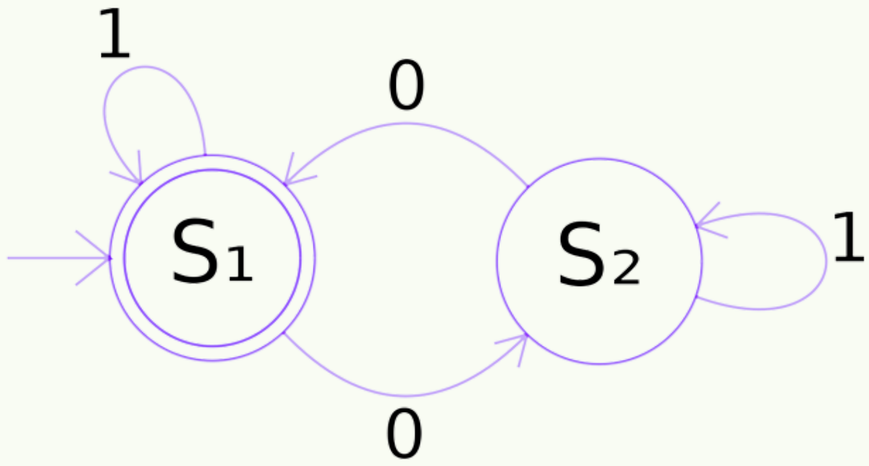
Autômatos Finitos Determinístico

Conceito

- Máquina de estados finitos, utilizada para processar e reconhecer sequências de símbolos em uma linguagem formal

Quíntupla

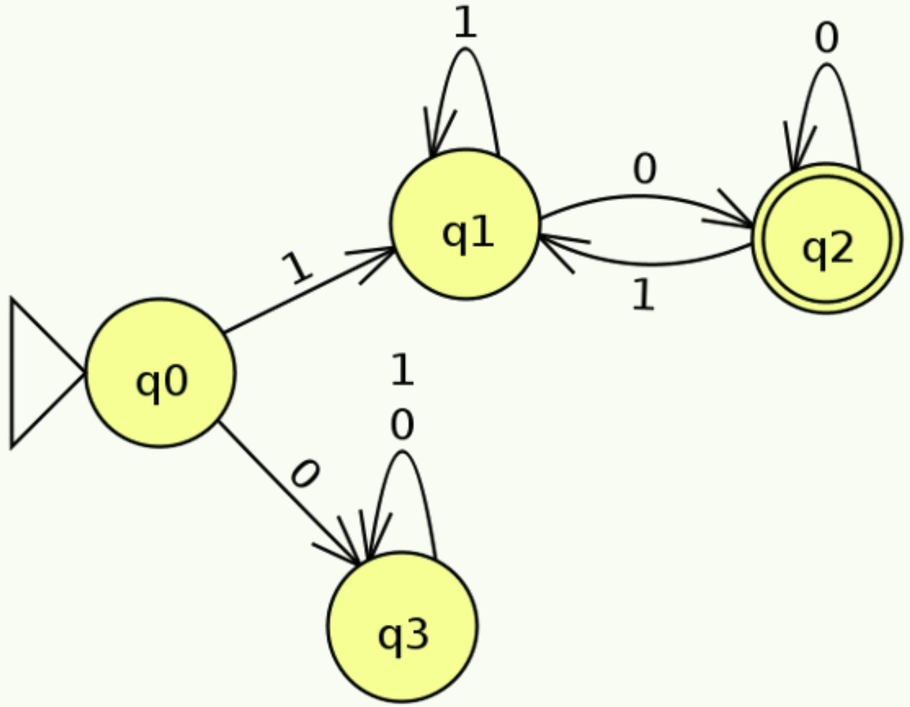
- Q = conjuntos de estados
- ∑ = alfabeto (Sigma)
- δ = conjunto de transições (Delta)
- Q0 = Estado inicial
- F = estado final (ou estados finais)



Autômatos Finitos Determinístico

Exemplo 1: Considerando o alfabeto Σ = {0,1} construa um AFD que aceite a seguinte descrição

- L = {ω: ω inicia com 1 e termina com 0}



Representando Expressões Regulares

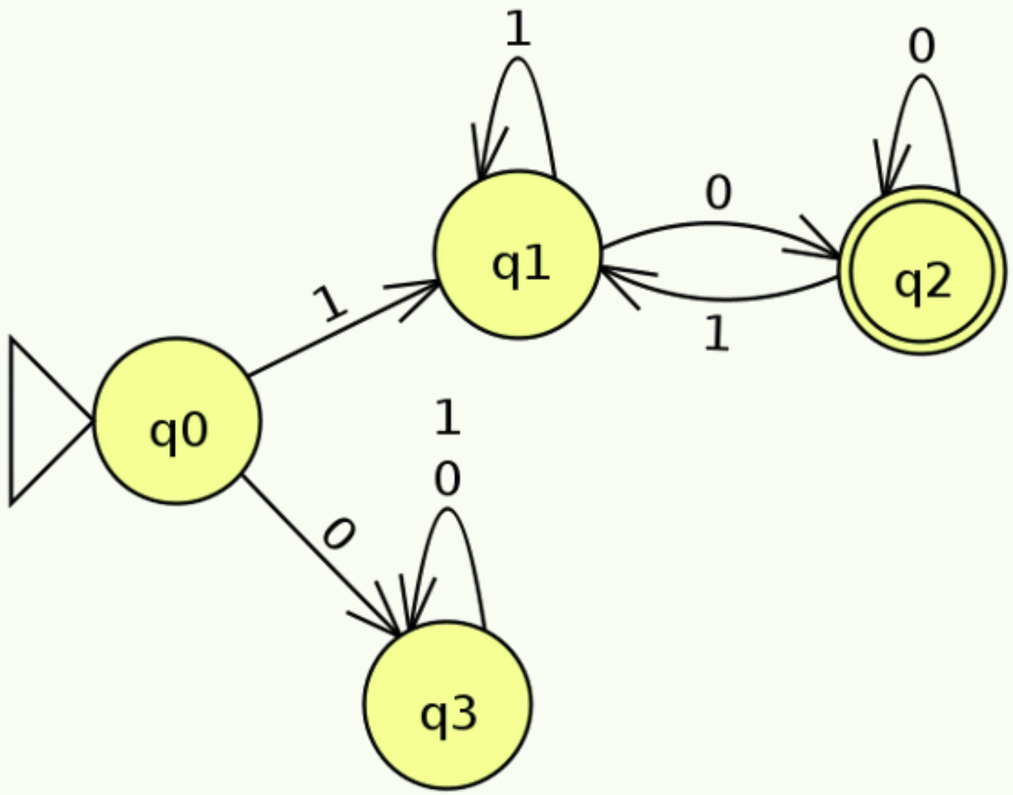
Exemplo 1: Considerando o alfabeto Σ = {0,1} escreva uma ER que aceite a seguinte descrição:

- L = {ω: ω inicia com 1 e termina com 0}

Solução usual: (11*00*)(11*00*)*

Representando Expressões Regulares

Solução usual: (11*00*)(11*00*)*

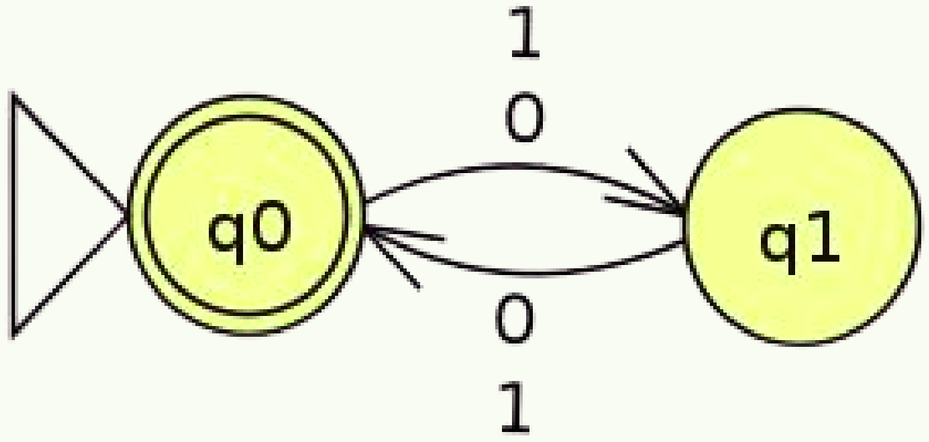


Estado	Expressão
q0, q1	(**1**1*00*)(11*00*)*
q1, q1	(1**1***00*)(11*00*)*
q1, q2	(11***0**0*)(11*00*)*
q2, q2	(11*0**0***)(11*00*)*
q2, q1	(11*00*)(**1**1*00*)*

Autômatos Finitos Determinístico

Exemplo 2: Considerando o alfabeto Σ = {0,1} construa um AFD que aceite a seguinte descrição

- L = {ω: ω tem comprimento par}



Representando Expressões Regulares

Exemplo 2: Considerando o alfabeto ∑ = {0,1} construa um AFD que aceite a seguinte descrição

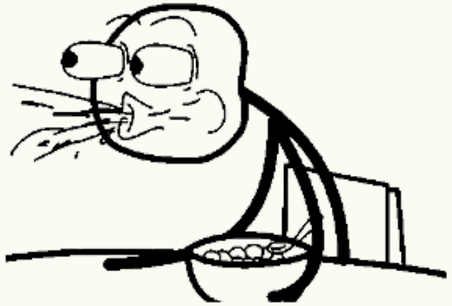
- L = {ω: ω tem comprimento par}

Solução usual: ((1+0)(1+0))*

Obs: Cadeia vazia é aceita

Representação em Python

^([0-9]{3}\.?){2}([0-9]{3}-?)(\d\d)$



Expressões Regulares com Python

- As expressões regulares são suportadas em Python através da biblioteca padrão chamada **re**

```
import re
```

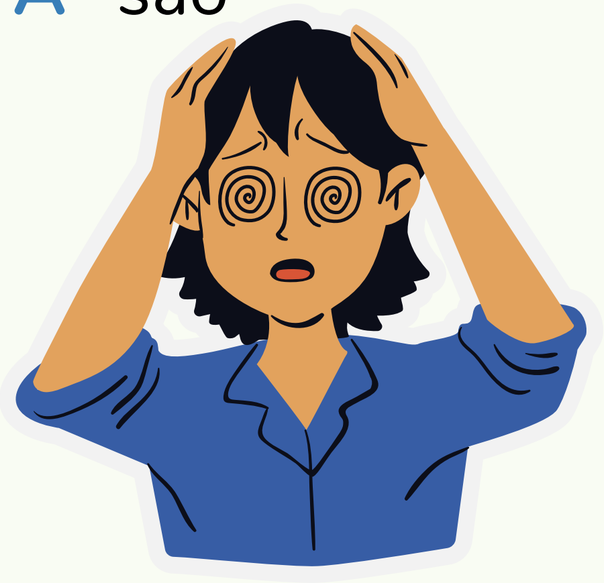
- Utilizamos expressão r'<expressao>' para criar uma string “raw” (bruta) em Python. É uma forma de representar uma string sem que caracteres especiais sejam interpretados

Exemplo: r'hello'

Expressões Regulares com Python

- Qualquer caractere presente na expressão será tratado de forma literal, seja espaços e quebras de linha
- Por exemplo, a letra **"a"** na expressão, coincide apenas com a letra **"a"**. Já a letra **"A"** irá coincidir somente com **"A"** são interpretados literalmente

r'hello' ≠ r' hello' ≠ r'hello ' ≠ r' hello '



Metacaracteres

- Metacaracteres são caracteres especiais usados em expressões regulares que têm significados especiais e são utilizados para definir padrões de busca mais complexos
- Os metacaracteres mais comuns em expressões regulares são:

Asterisco: *****
Chave: **{ }**
Colchetes: **[ ]**
Circunflexo: **^**

Cifrão: **$**
Barra vertical: **|**
Parênteses: **( )**
Barra inversa: ****

Operadores de quantidade: Asterisco

Asterisco (*): zero ou mais ocorrências

Exemplo: Considere a linguagem com ∑ = {a,b}, gere uma expressão regular que aceite toda palavra que comece com "a" seguido de zero ou mais "b"

Representação Usual

Solução: ab*

Representação em Python

Solução: r'ab*'

Palavras aceitas: a, ab, abb, abbb ...

Operadores de quantidade: Asterisco

- Podemos utilizar o ***** para representar **uma** ou **mais** vezes

Exemplo: Considere a linguagem com ∑ = {a,b}, gere uma expressão regular que aceite toda palavra que comece com "a" seguido obrigatoriamente de pelo menos um "b"

Representação Usual

Solução: abb*

Representação em Python

Solução: r'abb*'

Palavras aceitas: ab, abb, abbb, abbbb ...

Exemplo

texto = '''esta me **d**esejan**d**o um bom **diii**aaa, ou quer **diiii**zzzeeerrr que eh um bom **di**a quer eu queira ou nao; ou quer dizer que voce se sente bem neste **di**a; ou que este eh um bom **dii**aaa para ser bom?'''

```
regex = r'di*'
```

Palavras aceitas: d, d, diii, diiii, di, di, di, dii

Operadores de quantidade: Chaves

- **Colchetes {}:** utilizado como range para repetições
- **{m,n}:** a repetição começa em m e termina em n

Considere a linguagem com ∑ = {a,b}

Exemplo 1: `r'a{1,3}'`

Cadeias de retorno: a, aa, aaa

Exemplo 2: `r'a{3}'`

Cadeias de retorno: aaa

Exemplo

texto ="""esta me desejando um bom **diii**aaa, ou quer **diii**izzzeeerrr que eh um bom **di**a quer eu queira ou nao; ou quer **di**zer que voce se sente bem neste **di**a; ou que este eh um bom **diii**aaa para ser bom?"'

```
regex = r'di{1,3}'
```

Palavras aceitas: diii, diii, di, di, di, diii

Metacaractere: colchetes

- **Chaves [ ]:** significa conjunto de caracteres

Exemplo 1:

```
regex = r'[abc]'
```

Lê-se: Qualquer caractere **a**, **b**, **OU c**

obs: Podemos escrever **[a-c]**, que usa um range para expressar o mesmo conjunto de caracteres

Exemplo 2:

```
r'[a-z]'
```

RE usada encontrar apenas letras minúsculas no alfabeto

Metacaractere: colchetes

- Podemos utilizar * em conjuntos

Exemplo 1:

```
regex = r'[abc]*'
```

A RE irá encontrar qualquer sequência de caracteres que contenha apenas as letras **a, b** ou **c**, **zero** ou **mais** vezes

Cadeias de retorno: abacaba, abb, aaaa ...

Metacaractere: Início e Fim

- **Circunflexo ^:** Corresponde ao início de linha

Exemplo 1:

```
regex = r'^may'
```

frase1 = '''**may** the force be with you'''

frase2 = '''with you, **may** the force be'''

Metacaractere: Início e Fim

- **Cifrão $:** Corresponde ao fim de uma linha, que tanto é definido como o fim de uma string, ou qualquer local seguido por um caractere de nova linha

Exemplo 1:

```
regex = r'}$'
```

frase1 = '''{block}'''

frase2 = '''{block} '''

frase 3 = '''{block}\n'''

Operadores de união: barra vertical

- **Barra vertical (|):** Alternância, ou o operador "ou"

Exemplo: Considere a linguagem com ∑ = {a,b}

Representação Usual

Solução: a+b

O símbolo "+" significa "ou"

Representação em Python

Solução: `r'a|b'`

`r'[ab]'`

regex = `r'bilbo|smeagol'`

A expressão irá corresponder tanto **bilbo** quanto **smeagol**

Meta caracteres: Agrupamento

- **Parênteses: ( )** agrupam as expressões contidas dentro deles
- Quantificadores repetem conteúdo de grupos, ***** ou **{m,n}**
- Parênteses capturam texto correspondente a grupo.

```
regex = r'(ab)*'
```

Palavras aceitas: ab,abab, abababab...

Meta caracteres: Agrupamento

texto = '''esta me desejando um bom diiiaaa, ou quer diiiizzzeeeerrr que eh um bom **di**a quer eu queira ou nao; ou quer dizer que voce se sente bem neste **di**a; ou que este eh um bom diiiaaa para ser bom?'''

```
regex = r'(di)a'
```

Resultado: di, di

Não captura de grupos

- **Grupo não capturado (?:) :** grupo para denotar uma parte de uma expressão regular, sem recuperar o conteúdo do grupo

texto = '''abc'''

```
regex = r'([abc])[abc]'
```

Resultado: a

```
regex = r'(?:[abc])[abc]'
```

Resultado: ab

Não captura de grupos

texto = '''esta me desejando um bom **diiia**aa, ou quer diiiizzzeeeerrr que eh um bom **dia** quer eu queira ou nao; ou quer dizer que voce se sente bem neste **dia**; ou que este eh um bom **diiia**aa para ser bom?'''

```
regex = r'(?:di|diii)a'
```

Resultado: diiia, dia, dia, diiia

A bendita barra invertida

- `\`: indicar formas especiais ou para permitir que caracteres especiais sejam usados sem invocar o seu significado especial

```
regex = r'[ ]'
```

Resultado: ERROR

```
regex = r'()'
```

Resultado: , , , , , …

```
regex = r'\[\ ]'
```

Resultado: [ ]

```
regex = r'\(\)'
```

Resultado: ( )

Não captura de grupos

texto = '''esta me desejando um bom **(diiia)**aa, ou quer diiiizzzeeeerrr que eh um bom **(dia)** quer eu queira ou nao; ou quer dizer que voce se sente bem neste (di)a; ou que este eh um bom diiiaaa para ser bom?'''

```
regex = r'(?:\(di|\(diii)a\)'
```

Resultado: (diiia), (dia)

Exemplos

Linguagem	Expressão Regular em Python
CPF	`r'[0-9]{3}\.[0-9]{3}\.[0-9]{3}-[0-9]{2}'`
Datas no formato DD/MM/AAAA	`r'\d{2}/\d{2}/\d{4}'`
Números de telefone	`r'\(\d{3}\) \d{3}-\d{4}'`
Palavras em negrito	`r'\*\*(.*?)\*\*'`
Email	`r'\w+(?:[.\-+!%]\w+)*@\w+(?:[.\-]\w+)+'`
Endereços IP	`r'\d+.\d+.\d+.\d'`

Unicode

- **\d**: qualquer dígito decimal, equivalente à **[0-9]**
- **\D**: qualquer caractere não-dígito, equivalente à **[^0-9]**
- **\s:** qualquer caractere espaço-em-branco, equivalente à **[\t\n\r\f\v]**
- **\S**: qualquer caractere não-espaço-branco, equivalente **[^\t\n\r\f\v]**
- **\w**: qualquer caractere alfanumérico, equivalente à **[a-zA-Z0-9_]**
- **\W:** qualquer caractere não-alfanumérico, similar à **[^a-zA-Z0-9_]**

Unicode: \d e \D

texto = '''O que o matemático disse para o índio brasileiro?
8π
E o que ele disse quanto não teve resposta?
18π !'''

```
regex = r'\d'
```

Resultado: 8, 1, 8

```
regex = r'\D''
```

Resultado: O, , q, u, e, , o, , m, a, t, e, m, á, t, i, c, o, , d, i, s, s, e, , p, a, r, a, , o, , í, n, d, i, o, , b, r, a, s, i, l, e, i, r, o, ?, \n, π, \n, E, , o, , q, u, e, , e, l, e, , d, i, s, s, e, , q, u, a, n, t, o, , n, ã, o, , t, e, v, e, , r, e, s, p, o, s, t, a, ?, \n, π, , !

Unicode: \w

texto = '''esta me desejando um bom diiiaaa, ou quer diiiizzzeeeerrr que eh um bom dia quer eu queira ou nao; ou quer dizer que voce se sente bem neste dia; ou que este eh um bom diiiaaa para ser bom?'''

```
regex = r'\w\w*'
```

Resultado: esta, me, desejando, um, bom, diiiaaa, ou, quer, diiiizzzeeeerrr, que, eh, um, bom, dia, quer, eu, queira, ou, nao, ou, quer, dizer, que, voce, se, sente, bem, neste, dia, ou, que, este, eh, um, bom, diiiaaa, para, ser, bom

Compilando Expressões Regulares

As expressões regulares são compiladas em objetos padrão, que têm métodos para várias operações, tais como a procura por padrões de correspondência ou realizar substituições de strings

```
regex = re.compile(r'ab*')
regex.function()
```

Function() representa qualquer método que ainda veremos adiante...

Executando Comparações

MÉTODO/ATRIBUTO	PURPOSE
match()	Determina se a RE combina com o início da string.
search()	Varre toda a string, procurando qualquer local onde esta RE tem correspondência.
findall()	Encontra todas as substrings onde a RE corresponde, e as retorna como uma lista.
finditer()	Encontra todas as substrings onde a RE corresponde, e as retorna como um iterador.

Método: match()

texto = '''esta me desejando um bom diiiaaa, ou quer diiiizzzeeerrr que eh um bom dia quer eu queira ou nao; ou quer dizer que voce se sente bem neste dia; ou que este eh um bom diiiaaa para ser bom?'''

```
regex = re.compile(r'd\w*')
regex.match(texto)
```

```
re.match(r'd\w*', texto)
```

Resultado: None

```
regex = re.compile(r'e\w*')
regex.match(texto)
```

Resultado: <re.Match object; span=(0, 4), match='esta'>

Método: search()

texto = '''esta me desejando um bom diiiaaa, ou quer diiiizzzeeerrr que eh um bom dia quer eu queira ou nao; ou quer dizer que voce se sente bem neste dia; ou que este eh um bom diiiaaa para ser bom?'''

```
regex = re.compile(r'd\w*')
regex.search(texto)
```

Resultado: <re.Match object; span=(8, 17), match='desejando'>

Método: findall()

texto = '''esta me desejando um bom diiiaaa, ou quer diiiizzzeeeerrr que eh um bom dia quer eu queira ou nao; ou quer dizer que voce se sente bem neste dia; ou que este eh um bom diiiaaa para ser bom?'''

```
regex = re.compile(r'd\w*')
regex.findall(texto)
```

Resultado: [desejando, diiiaaa, diiiizzzeeeerrr, dia, dizer, dia, diiiaaa]

Método: finditer()

texto = '''esta me desejando um bom diiiaaa, ou quer diiiizzzeeeerrr que eh um bom dia quer eu queira ou nao; ou quer dizer que voce se sente bem neste dia; ou que este eh um bom diiiaaa para ser bom?'''

```
regex = re.compile(r'd\w*')
regex.finditer(texto)
```

Resultado: <callable_iterator object at 0x7f478a2d55a0>

Método: finditer()

```
regex = re.compile(r'd\w*')
for  i  in  regex.finditer(texto):
  print(i)
```

```
<re.Match object; span=(8, 17), match='desejando'>
<re.Match object; span=(25, 32), match='diiiaaa'>
<re.Match object; span=(42, 56), match='diiiizzzeeerrr'>
<re.Match object; span=(71, 74), match='dia'>
<re.Match object; span=(106, 111), match='dizer'>
<re.Match object; span=(140, 143), match='dia'>
<re.Match object; span=(167, 174), match='diiiaaa'>
```

Sinalizadores de Compilação

- **DOTALL, S**

Faz com que o “.” corresponder a qualquer coisa, incluindo novas linhas

- **IGNORECASE, I**

Faz combinações sem diferenciar maiúsculo de minúsculo

- **MULTILINE, M**

Correspondência multilinha, afetando ^ e $

DOTALL, S

texto = '''esta me desejando um bom diiiaaa, ou quer diiiizzzeeerrr que eh um bom dia quer eu queira ou nao; ou quer dizer que voce se sente bem neste dia; ou que
este eh um bom diiiaaa para ser bom?'''

```
regex = r'.'
```

Resultado: e, s, t, a, , m, e, , d, e, s, e, j, a, n, d, o, , u, m, , b, o, m, , d, i, i, i, a, a, a, ,, , o, u, , q, u, e, r, , d, i, i, i, i, z, z, z, e, e, e, r, r, r, , q, u, e, , e, h, , u, m, , b, o, m, , d, i, a, , q, u, e, r, , e, u, , q, u, e, i, r, a, , o, u, , n, a, o, ;, , o, u, , q, u, e, r, , d, i, z, e, r, , q, u, e, , v, o, c, e, , s, e, , s, e, n, t, e, , b, e, m, , n, e, s, t, e, , d, i, a, ;, , o, u, , q, u, e, , e, s, t, e, , e, h, , u, m, , b, o, m, , d, i, i, i, a, a, a, , p, a, r, a, , s, e, r, , b, o, m, ?

DOTALL, S

texto = '''
esta me desejando um bom diiiaaa, ou quer diiiizzzeeerrr que eh um bom dia quer eu queira ou nao; ou quer dizer que voce se sente bem neste dia; ou que este eh um bom diiiaaa para ser bom?'''

```
regex = re.compile(r'.', flags=re.S)
```

Resultado: \n, e, s, t, a, , m, e, , d, e, s, e, j, a, n, d, o, , u, m, , b, o, m, , d, i, i, i, a, a, a, ,, , o, u, , q, u, e, r, , d, i, i, i, i, z, z, z, e, e, e, r, r, r, , q, u, e, , e, h, , u, m, , b, o, m, , d, i, a, , q, u, e, r, , e, u, , q, u, e, i, r, a, , o, u, , n, a, o ...

IGNORECASE, I

texto = '''Esta me desejando um bom Diiiaaa, ou quer Diiiizzzeeerrr que eh um bom dia quer eu queira ou nao;
ou quer dizer que voce se sente bem neste dia; ou que este eh um Bom diiiaaa para ser Bom?'''

```
regex = re.compile(r'[A-Z][a-z]*')
```

Resultado: Esta, Diiiaaa, Diiiizzzeeerrr, Bom

IGNORECASE, I

texto = '''Esta me desejando um bom Diiiaaa, ou quer Diiiizzzeeerrr que eh um bom dia quer eu queira ou nao;
ou quer dizer que voce se sente bem neste dia; ou que este eh um Bom diiiaaa para ser Bom?'''

```
regex = re.compile(r'[a-z][a-z]*', flags=re.I)
```

Resultado: Esta, me, desejando, um, bom, Diiiaaa, ou, quer, Diiiizzzeeerrr, que, eh, um, bom, dia, quer, eu, queira, ou, nao, ou, quer, dizer, que, voce, se, sente, bem, neste, dia, ou, que, este, eh, um, Bom, diiiaaa, para, ser, Bom

MULTILINE, M

texto = '''esta me desejando um bom diiiaaa, ou quer diiiizzzeeeerrr que eh um bom dia quer
eu queira ou nao; ou quer dizer que voce se sente bem neste dia; ou que este eh um bom diiiaaa para ser bom?'''

```
regex = r'^e\w*'
```

Resultado: esta

```
regex = re.compile(r'^e\w*', flags=re.M)
```

Resultado: esta, eh, eu, este

Gulosos versus não Gulosos

Ao repetir uma expressão regular, como em a*, a ação resultante é consumir o tanto do padrão quanto possível.

texto = '''<html><head><title>Title</title>'''

```
regex = r'<.*>'
```

Resultado esperado: <html>, <head>, <title>, </title>

Resultado Obtido: <html><head><title>Title</title>

Gulosos

texto = '''esta me desejando um bom diiiaaa, ou quer diiiizzzeeeerrr que eh um bom dia quer
eu queira ou nao; ou quer dizer que voce se sente bem neste dia; ou que este eh um bom diiiaaa para ser bom?'''

```
regex = re.compile(r'e.*m', flags=re.S)
```

Resultado: esta me desejando um bom diiiaaa, ou quer diiiizzzeeeerrr que \neh um bom dia quer \neu queira ou nao; ou quer dizer que voce se sente bem neste dia; ou que \neste eh um bom diiiaaa para ser bom

Não Gulosos

texto = '''esta me desejando um bom diiiaaa ou quer diiiizzzeeeerrr que eh um bom dia quer
eu queira ou nao; ou quer dizer que voce se sente bem neste dia; ou que este eh um bom diiiaaa para ser bom?'''

```
regex = re.compile(r'e.*?m', flags=re.S)
```

Resultado: esta m, e desejando um, er diiiizzzeeeerrr que \neh um, er \neu queira ou nao; ou quer dizer que voce se sente bem, este dia; ou que \neste eh um, er bom

Lookahead

É uma técnica usada em expressões regulares para especificar uma condição que deve ser satisfeita sem fazer parte do resultado correspondente.

- Existem dois tipos de lookahead em expressões regulares:

1. Afirmação lookahead positiva
2. Afirmação lookahead negativa

Lookahead positiva

(?=...)

- Retorna sucesso se a expressão regular informada, aqui representada por **...**, corresponde com o conteúdo da localização atual, e retorna falha caso contrário.
- Uma vez que a expressão informada tenha sido testada, o mecanismo de correspondência não faz qualquer avanço; o resto do padrão é tentado no mesmo local de onde a afirmação foi iniciada.

Lookahead positiva

texto = '''esta me desejando um bom diiiaaa, ou quer diiiizzzeeerrr que eh um bom dia quer eu queira ou nao; ou quer dizer que voce se sente bem neste dia; ou que este eh um bom diiiaaa para ser bom?'''

```
regex = r'\w\w*(?=;)'
```

Resultado: não, dia

```
regex = r'\w\w*(?=\?)'
```

Resultado: bom

Lookahead negativa

(?!...)

- É o oposto da afirmação positiva; será bem-sucedida se a expressão informada não corresponder com o conteúdo da posição atual na string

```
regex = r'{exp}(?!...)'
```

Obs: se `(?!...)` não ocorrer, retornamos a expressão desejada `{exp}`

Lookahead negativa

texto = '''esta me desejando um bom diiiaaa, ou quer diiiizzzeeerrr que eh um bom dia quer eu queira ou nao; ou quer dizer que voce se sente bem neste dia; ou que este eh um bom diiiaaa para ser bom?'''

```
regex = r'\w\w*(?!;)'
```

Resultado: esta, me, desejando, um, bom, diiiaaa, ou, quer, diiiizzzeeerrr, que, eh, um, bom, dia, quer, eu, queira, ou, nao, ou, quer, dizer, que, voce, se, sente, bem, neste, dia, ou, que, este, eh, um, bom, diiiaaa, para, ser, bom

Lookbehind

Permite verificar se um determinado padrão de caracteres está presente antes da posição atual na string, sem realmente incluir essa parte correspondente na correspondência resultante

- Existem dois tipos de lookbehind em expressões regulares:

1. Lookbehind positiva
2. Lookbehind negativa

Lookbehind Positivo

(?<=...)

Retorna sucesso se a expressão regular informada é precedida por um padrão, aqui representada por **...**, caso contrário retorna falha

```
regex = r'(?<=...){exp}'
```

A expressão permite verificar se {exp} é precedido por (?<=...) sem incluí-lo na correspondência. Se a expressão é confirmada, então a {exp} é retornada

Lookbehind positiva

texto = '''esta me desejando um bom dia, ou quer dizer que é um bom dia quer eu queira ou não;ou que você se sente bem neste dia;ou que é um bom dia para ser bom?'''

```
regex = r'(?<=;)\w\w*'
```

Resultado: ou, ou

A expressão regular busca por qualquer sequência de caracteres que contenha um ou mais caracteres alfanuméricos **\w** que seja precedida pelo caractere **;**

Lookbehind negativa

?<!...)

Retorna sucesso se a expressão regular informada não é precedida por um padrão, aqui representada por ..., caso contrário retorna falha

```
regex = r'?<!{exp}'
```

O lookbehind em expressões regulares possibilita verificar se um padrão específico {exp} é precedido por (?<=...) sem incluí-lo na correspondência. Se a expressão é confirmada, então a {exp} é retornada

Lookbehind negativa

texto = '''esta me desejando um bom dia, ou quer dizer que é um bom dia quer eu queira ou não;ou que você se sente bem neste dia;ou que é um bom dia para ser bom?'''

```
regex = r'(?<!;)\w+'
```

Resultado: esta, e, esejando, m, om, ia, u, uer, izer, ue, m, om, ia, uer, u, ueira, u, ão, ou, ue, ocê, e, ente, em, este, ia, ou, ue, m, om, ia, ara, er, om

A expressão regular busca por qualquer sequência de caracteres que contenha um ou mais caracteres alfanuméricos **\w+** que seja precedida por um espaço


DID SOMEONE SAY
REGULAR EXPRESSIONS

REFERENCES

REFERENCES 1
https://docs.python.org/pt-br/3.8/howto/regex.html