Aplicações de Expressões Regulares: Máscaras de Validação e Reconhecimento de Arranjos Familiares

Juliana A. S. Feio[1]

[1]Instituto de Ciências Exatas e Naturais – Universidade Federal do Pará (UFPA)
Belém – PA – Brasil

juliana.feio@icen.ufpa.br

Abstract. *This paper consists in a technical report that presents the application of regular expressions in form fields validation, that there usually in manifold applications for web, mobile and desktop. Moreover, regular expressions were built to the languages recognize arrangements family according by criteria constituted by different rules combinations related to individuals age and sex.*

Resumo. *Este trabalho consiste em um relatório técnico que apresenta a aplicação de expressões regulares na validação de campos de preenchimento em formúlários, que comumente se fazem presentes em diversas aplicações para web, mobile e desktop. Além disso, foram construídas expressões regulares para o reconhecimento de linguagens que representam arranjos familiares de acordo com critérios constituídos por combinações de diferentes regras em relação a idade, sexo e quantidade de indivíduos.*

1. Introdução

A Teoria das Linguagens Formais foi originalmente desenvolvida na década de 1950, com o objetivo de desenvolver teorias relacionadas com as linguagens naturais, no entanto foi verificado que esta teoria era importante para o estudo de linguagens artificiais, em especial na área de Ciência da Computação.[Menezes 1998]

1.1. Linguagens formais

Uma linguagem formal é um conjunto, finito ou infinito, de cadeias de comprimento finito formadas pela concatenação de símbolos pertencentes a um alfabeto finito e não vazio. A concatenação de símbolos segue a regra determinada pela linguagem, a qual é construída utilizando um conjunto de operações como fechamentos, operações lógicas e operações de conjuntos. [Ramos 2009]

1.2. Fechamentos

Uma relação (binária) é um subconjunto originado de um produto cartesiano entre dois conjuntos (AXB), onde o conjunto mais à esquerda é denotado domínio dessa relação e o conjunto mais à direita é denotado contradomínio dessa relação.[Menezes 1998]

Neste trabalho são utilizados dois tipos de fecho da relação AXA (onde o domínio e o contradomínio coincidem) comumente usados na construção de expressões regulares, os quais são fecho transitivo, e fecho transitivo e reflexivo.

Seja R uma relação em A (AXA), o fecho de R em relação ao conjunto de propriedades (transitiva, reflexiva), denominado Fecho Transitivo e Reflexivo de R e denotado por R*, [Menezes 1998] é o conjunto de todas as relações definíveis sobre AXA, também definido por $2^{A\times A}$.[Ramos 2009]

1.3. Expressões Regulares

Toda Linguagem Regular pode ser descrita por uma expressão simples, denominada Expressão Regular. Trata-se de um formalismo gerador, pois expressa como construir (gerar) as palavras da linguagem. É uma maneira de representar uma linguagem formada pela justaposição dos símbolos do alfabeto separados pelos símbolos de união ou concatenação. [Menezes 1998]

2. Materiais e Métodos

Este relatório técnico apresenta a aplicação de expressões regulares na validação de campos de preenchimento em formulários que se fazem presentes em diversas aplicações de software. Além disso, foram construídas expressões regulares para o reconhecimento de linguagens que representam arranjos familiares de acordo com regras em relação a idade, sexo e quantidade de indivíduos.

Nas subseções a seguir são especificados os materiais e métodos utilizados a fim de documentar este experimento para replicações e análises posteriores.

2.1. Anaconda e Jupyter Notebook

Anaconda é uma distribuição das linguagens de programação Python e R para computação científica, que visa simplificar o gerenciamento e implantação de pacotes. A distribuição contém pacotes muito utilizados em machine learning, análise e ciência de dados. [Anaconda 2021]

Entre outros recursos, Anaconda também conta com a aplicação Jupyter Notebook para criar e compartilhar documentos computacionais. O Projeto Jupyter é um projeto de código aberto sem fins lucrativos, nascido do Projeto IPython em 2014, à medida que evoluiu para dar suporte à ciência de dados interativa e à computação científica em todas as linguagens de programação. [Jupyter 2020]

2.2. A biblioteca *RE*

As expressões regulares foram construídas na linguagem Python usando a biblioteca *RE*, também conhecida como *Regex* (de *Regular Expressions*), importada como ”re”para o uso desta na programação de expressões regulares.

Este módulo fornece operações de correspondência de expressões regulares, suportando strings de 8 bits e Unicode. O padrão e as strings, sendo processadas, podem conter bytes nulos e caracteres fora do intervalo ASCII. [Van Rossum 2021]

As expressões regulares podem conter caracteres especiais, como parênteses, colchetes, chaves, e símbolos especiais; e comuns, como letras e números. [Van Rossum 2021] Tais caracteres quando combinados e concatenados formam as expressões regulares.

2.3. Construção das expressões regulares

A seguir são descritas as regras de concatenação dos símbolos, as quais servem de base para a construção das expressões regulares que as representam.

Questão 1. Considerando os alfabetos:

Σ = {a,b,c,d,e, ... ,z}

Γ = {A,B,C,D,E, ... , Z}

N = {0,1,2,3,4,5,6,7,8,9}

Definir máscaras de validação para os campos de formulário com as regras a seguir:

Nome, nome do meio e sobrenome

1. Nome, nome do meio e sobrenome devem vir separados por um espaço apenas

2. O nome do meio é opcional

3. Nome e sobrenome devem ser ambos não vazios

4. Não deve aceitar caracteres especiais ou números

5. O primeiro símbolo do nome e sobrenome, e do nome do meio se existir, deve ser do alfabeto Γ e os outros símbolos devem ser do alfabeto Σ.

Linguagem:

$L_1 = \{(\Gamma\Sigma^+)\_(\Gamma\Sigma^+\_)^i(\Gamma\Sigma^+)|i \leq 1\}$

E-mail

1. Sentenças devem conter um, e apenas um, símbolo "@"

2. Excetuando o símbolo "@", as sentenças possuem apenas símbolos de Σ

3. Sentenças não devem começar com o símbolo "@"

4. Sentenças devem terminar com a subcadeia ".com.br" ou ".br"

5. Sentenças devem ter, pelo menos, um símbolo de Σ entre o símbolo "@" e a subcadeia ".com.br" ou a subcadeia ".br".

Linguagem:

$L_2 = \{\Sigma^+@\Sigma^+(.com.br|br)\}$

Senha

1. Sentenças podem conter símbolos de $\Sigma \cup \Gamma \cup N$

2. Sentenças devem ter pelo menos um símbolo de e outro de N

3. Sentenças devem ter comprimento igual a 8

Linguagem:

$L_3 = \{(\Gamma^+ N^+ \Sigma^*)^i |\ i = 8\}$

CPF

1. Sentenças devem ter o formato xxx.xxx.xxx-xx, onde x $\in N$

Linguagem:

$L_4 = \{N^i.N^i.N^i - N^j |\ i = 3,\ j = 2\}$

Telefone

1. Sentenças devem estar no formato (xx) 9xxxx-xxxx

Linguagem:

$L_5 = \{'('\ N^2\ ')'\_9N^4 - N^4\}$

Data e Hora

1. Sentenças devem ter o formato dd/mm/aaaa hh:mm:ss, onde d, m, a, h, m, s $\in N$

Linguagem:

$L_6 = \{(0\ a\ 9)^2/(0\ a\ 9)^2/(0\ a\ 9)^4\_(00:00:00|00:00:(0\ a\ 5)(0\ a\ 9)|00:(0\ a\ 5)(1\ a\ 9):(0\ a\ 5)(0\ a\ 9)|00:(1\ a\ 5)(0\ a\ 9):(0\ a\ 5)(0\ a\ 9)|0(1\ a\ 9):(0\ a\ 5)(0\ a\ 9):(0\ a\ 5)(0\ a\ 9)|1(0\ a\ 9):(0\ a\ 5)(0\ a\ 9):(0\ a\ 5)(0\ a\ 9)|2(0\ a\ 3):(0\ a\ 5)(0\ a\ 9):(0\ a\ 5)(0\ a\ 9))\}$

Número real com ou sem sinal

1. Sentenças devem começar com um dos símbolos do alfabeto { +, -, ε}
2. Em seguida, as sentenças devem ter, pelo menos, um símbolo do alfabeto N
3. Em seguida, as sentenças devem ter, exatamente, um símbolo separador "."
4. Em seguida, as sentenças devem finalizar com, pelo menos, um símbolo de N
5. Exceção: números sem a parte fracionária também devem ser aceitos

Linguagem:

$L_7 = \{(+\|-)^* N^+ (.N)^*\}$

Questão 2. Considerando o alfabeto:

$\Sigma = \{H, M, h, m\}$

Onde:

◦ H representa um homem;

◦ M representa uma mulher;

◦ h representa um filho do sexo masculino (natural ou adotado);

◦ m representa uma filha do sexo feminimo (natural ou adotado);

◦ A posição relativa de uma letra em relação às demais indica a idade relativa daquele membro da família em relação aos demais (os mais novos estão sempre à direita).

Definir expressões regulares para cada proposição a seguir:

a) Casais heterossexuais mais velhos que os filhos com pelo menos duas filhas mulheres, ou pelo menos um filho homem, ou ainda pelo menos dois filhos homens e uma filha mulher.

Linguagem:

$L_8 = (HM|MH)((mm(mh)^*) \mid ((mh)^*mm) \mid (m(mh)^*m)) \mid ((h(mh)^*) \mid$ ((mh)$^*h$)) | $((h(mh)^*hm) \mid (hh(mh)^*m) \mid (hhm(mh)^*) \mid ((mh)^*hhm) \mid$ (m(mh)$^*hh$) | $(mh(mh)^*h) \mid (mhh(mh)^*) \mid ((mh)^*mhh) \mid (h(mh)^*mh) \mid$ (hm(mh)$^*h$) | $(hmh(mh)^*) \mid ((mh)^*hmh))$

b) Casais heterossexuais mais velhos que os filhos e com uma quantidade ímpar de filhas mulheres.

Linguagem:

$L_9 = \{(HM|MH)(h^*mh^*|(h^*mh^*(h^*mh^*m)^+)h^*)\}$

c) Casais heterossexuais mais velhos que os filhos, com a filha mais velha mulher e o filho mais novo homem.

Linguagem:

$L_{10} = \{(HM|MH)(m(hm)^*h)\}$

d) Casais homossexuais mais velhos que os filhos, com pelo menos seis filhos, em que os dois primeiros filhos formam um casal e os últimos também.

Linguagem:

$L_{11} = \{(HH|MM)(mh(hm)^imh)|(hm(hm)^ihm)|(mh(hm)^ihm)|(hm(hm)^imh)$, onde i $\geq 2\}$

e) Casais homossexuais mais velhos que os filhos, em que o sexo dos filhos é alternado conforme a ordem de nascimento.

Linguagem:

$L_{12} = \{(HH|MM)((h(\sim h))|m(\sim m))^+\}$

f) Casais homossexuais mais velhos que os filhos, com qualquer quantidade de filhos homens e mulheres, mas que não tiveram dois filhos homens consecutivos.

Linguagem:

$L_{13} = \{(HH|MM)((h(\sim h))|m)^*\}$

g) Arranjo de no mínimo x $\in N$ e no máximo y $\in N$, com x > 0 , y > 0 , e x $\leq$ y , de adultos (Hs ou Ms) mais velhos que os filhos, com qualquer quantidade de filhos homens e mulheres, mas que os três filhos mais novos não foram homens.

Linguagem:

$L_{14} = \{(H|M)^i((h|m)^* m^j|m^*),\ onde\ 1 < i < 7\ e\ j \geq 3\}$

3. Testes experimentais

Para testar as expressões regulares definidas acima a fim de descobrir se elas atendem, respectivamente, às regras definidas pelas linguagens, foram criadas uma série de listas contendo entre 10 e 15 cadeias que deverão ser reconhecidas (aceitas), ou seja, que pertencem à linguagem; e uma série de listas contendo entre 10 e 20 cadeias que não poderão ser reconhecidas, ou seja, que não pertencem à linguagem.

3.1. Linguagens em código Python

Os testes foram realizados utilizando a bibioteca *re* da linguagem de programação Python. Devido à dificuldade de visualização, a tabela abaixo apresenta apenas algumas das linguagens definidas acima e suas respectivas escritas em código. Todas as expressões regulares podem ser verificadas no código em anexo.

Tabela 1. Linguagens e suas respectivas escritas em código Python

Linguagem	Expressão Regular em Python
$\{\ (\Gamma\Sigma^+)\_(\Gamma\Sigma^+\_)^i(\Gamma\Sigma^+)\|i \leq 1\}$	[A-Z][a-z]+ ([A-Z][a-z]+)?[A-Z][a-z]+
$\{\ \Sigma^+@\Sigma^+(.com.br\|.br)\}$	[a-z]+@[a-z]+(.com.br||$.br$)
$\{\ (\Gamma^+ N^+ \Sigma^*)^i\|\ i = 8\}$	(?=.*[A-Z])(?=.*[0 a 9])[0 a 9a-zA-Z]{8}
$\{\ \mathrm{N}^i.N^i.N^i - N^j\|\ i = 3,\ j = 2\}$	[0-9]{3}[.][0-9]{3}[.][0-9]{3}[-][0-9]{2}$
$\{\ (\mathrm{N}^2)\_9N^4 - N^4\}$	[(][0-9]{2}[)] 9[0-9]{4}[-][0-9]{4} $
$\{(+\|\|-)^* N^+.^* N^*\}$	[+-]?[0-9]+([.][0-9]+)*
$\{(\mathrm{HM}\|\|MH)(m(hm)^* h)\}$	(HM$\|\|MH)(m[hm] * h)$
$\{\ (\mathrm{HH}\|\|MM)((h(\sim h))\|m(\sim m))^+\}$	(HH$\|\|MM)((h(?!h))\|\|m(?!m)) + \$$
$\{(\mathrm{HH}\|\|MM)((h(\sim h))\|m)^*\}$	(HH$\|\|MM)((h(?!h))\|m) * \$$

3.2. Cadeias testadas

As cadeias testadas foram criadas das mais variadas formas, seja para aceitação ou rejeição com base nas expressões regulares. Todos os testes mostraram que as expressões

regulares definidas neste trabalho atendem às regras solicitadas nas suas respecivas linguagens e tal afirmação pode ser verificada no anexo deste documento.

Para uma melhor visualização do código desenvolvido, um documento Jupyter Notebook foi elaborado com todas as questões listadas e respondidas, contando com um índice para uma melhor navegação no documento. Este arquivo está disponvel em $<$ github.com/julianaAuzier/teoria-da-computacao/blob/main/TC-expressoes-regulares.ipynb $>$.

4. Comentários

Atualmente o uso de expressões regulares inclui busca e substituição de texto em editores de texto, validação da informação dada em campos de preenchimento em formulários, e em mineração de textos as expressões regulares são também muito utilizadas.

Particularmente, este trabalho foi de grande aproveitamento para o aprendizado deste tema e da biblioteca *re* na linguagem Python.

4.1. Algumas expressões regulares

Existem linguagens em que podem parecer óbvias as expressões regulares que as representam, no entanto isto não é uma regra geral, como é o caso das questões abaixo.

◦ Data e Hora

Apesar de não ter limitações quanto aos números que compõem a data, há limitações dada na questão em relação a hora. Dessa forma, é necessário o mapeamento dos casos onde os números podem ter limites diferentes. Nesta linguagem, o primeiro caractere referente a hora pode ser 0, 1 ou 2; mas o segundo caractere possui limites de acordo com o primeiro caractere que aparece (não existe a hora 24, por exemplo, mas existe a hora 14).

◦ Uma quantidade ímpar de filhas mulheres

Neste caso vemos a necessidade da percepção do comportamento dos números. Com excessão ao número 1, todos os números ímpares são exatamente 2 unidades depois de outro número ímpar (1+2=3; 3+2=5; 5+2=7; etc.). Com este conhecimento é possível criar a expressão regular de forma simples, observando também que podem ter filhos homens entre as ”ímpares” filhas mulheres.

◦ O sexo dos filhos é alternado conforme a ordem de nascimento

Na definição da expressão regular neste documento é usado o símbolo $\sim$ para denotar ”não”. Neste caso, um filho homem (h) não pode ser seguido por outro filho homem (h), da mesma forma nos casos de filhas mulheres (m). A definição h ($\sim$ h), determina que na ocorrência de um h, em seguida não poderá haver outro h, e de forma análoga para a definição m ($\sim m$).

Referências

Anaconda (2021). Anaconda software distribution. https://docs.anaconda.com. Accessed: 2022-05-02.

Jupyter (2020). Jupyter notebook. https://jupyter.org. Accessed: 2022-05-02.

Menezes, P. B. (1998). *Linguagens formais e automatos*. Sagra-Dcluzzato.

Ramos, M. V. M. (2009). *Linguagens formais: Teoria, Modelagem e Implementação*. Artmed.

Van Rossum, G. (2021). *Regular Expression, release 3.10*. Python Software Foundation.

Anexos

Disponível em: <github.com/julianaAuzier/teoria-da-computacao/blob/main/TC-expressoes-regulares.ipynb>

Expressões regulares em Python: Funções e Resultados dos testes experimentais

1.1. Nome completo

```
1
def verificaNome(nome):
3     if re.match('[A-Z][a-z]+ ([A-Z][a-z]+ )?[A-Z][a-z]+',nome) == None:
          return 'Nome rejeitado'
5     else:
          verificacao = re.match('[A-Z]+[a-z]+ ([A-Z]+[a-z]+ )?[A-Z]+[a-z
              ]+',nome).group().split()
7         if verificacao == nome.split():
              return 'Nome aceito'
9         else:
              return 'Nome rejeitado'
11

13 nomesAceitos = ['Ada Lovelace',
                   'Alan Turing',
15                 'Stephen Cole Kleene',
                   'Juliana Auzier Seixas',
17                 'Ju Auzier',
                   'Ju Se',
19                 'Alan Mathison Turing',
                   'Stephen Cole',
21                 'Augusta Ada Byron',
                   'Augusta Ada']
23

   nomesRejeitados = ['1Alan',
25                    'Alan',
                      'Alan',
27                    'Alan Turing',
                      'Alan turing',
29                    'Juliana A Seixas',
                      'juliana Auzier Seixas',
31                    'Juliana S.',
                      'Juliana Auzier Seixas Feio',
33                    '',
                      'Ju ASF',
35                    'AAugsta Aa']


37

   print('Dever o ser aceitos:')
39 for i in nomesAceitos:
       print(verificaNome(i))
41

   print('\nDever o ser rejeitados:')
43 for i in nomesRejeitados:
       print(verificaNome(i))
```

Resultado:

```
Deverao ser aceitos:
Nome aceito
Nome aceito
Nome aceito
Nome aceito
Nome aceito
Nome aceito
Nome aceito
Nome aceito
Nome aceito
Nome aceito

Deverao ser rejeitados:
Nome rejeitado
Nome rejeitado
Nome rejeitado
Nome rejeitado
Nome rejeitado
Nome rejeitado
Nome rejeitado
Nome rejeitado
Nome rejeitado
Nome rejeitado
```

1.2. E-mail

```
def verificaEmail(email):
    if re.match('[a-z]+@[a-z]+(.com.br|.br)',email) == None:
        return 'Email rejeitado'
    else:
        verificacao = re.match('[a-z]+@[a-z]+(.com.br|.br)',email).
            group().split()
        if verificacao == email.split():
            return 'Email aceito'
        else:
            return 'Email rejeitado'


emailsAceitos = [ 'a@a.br',
                  'divulga@ufpa.br',
                  'a@a.com.br',
                  'email@email.br',
                  'ufpa@gmail.br',
                  'meuemail@mail.com.br',
                  'meuemail@mail.br',
                  'emaildealguem@servidor.br',
                  'emaildealguem@servidor.com.br',
                  'enderecodeemail@ufpa.br']

emailsRejeitados = ['@',
                    'a@.br',
                    '@a.br',
                    'T@teste.br',
                    'a@A.com.br',
                    '',
                    '@algo.com.br',
                    'email.ufpa@ufpa.br',
                    '1ema2il@ufpa.com.br',
                    'ufpa@gmail.com',
                    'ufpa@.com.br',
                    'email@email..br',
                    'email@email..com.br',
                    'email@ema.il.com.br'
                    'ema.il@email.com.br']


print('Deverao ser aceitos:')
for i in emailsAceitos:
    print(verificaEmail(i))

print('\nDeverao ser rejeitados:')
for i in emailsRejeitados:
    print(verificaEmail(i))
```

Resultado:

```
Deverao ser aceitos:
Email aceito
Email aceito
Email aceito
Email aceito
Email aceito
Email aceito
Email aceito
Email aceito
Email aceito
Email aceito

Deverao ser rejeitados:
Email rejeitado
Email rejeitado
Email rejeitado
Email rejeitado
Email rejeitado
Email rejeitado
Email rejeitado
Email rejeitado
Email rejeitado
Email rejeitado
Email rejeitado
Email rejeitado
Email rejeitado
Email rejeitado
```

1.3. Senha

```
def verificaSenha(senha):
    if re.match('(?=.*[A-Z])(?=.*[0-9])[0-9a-zA-Z]{8}',senha) == None:
        return 'Senha rejeitada'
    else:
        verificacao = re.match('(?=.*[A-Z])(?=.*[0-9])[0-9a-zA-Z]{8}',
            senha).group().split()
        if verificacao == senha.split():
            return 'Senha aceita'
        else:
            return 'Senha rejeitada'


senhasAceitas = ['518R2r5e',
                 'F123456A',
                 '1234567T',
                 'ropsSoq0',
                 'abcdfgX6',
                 '938S570z',
                 'Algo1234',
                 'Aceita00',
                 'Senha123',
                 'abABab12',
                 '101010OO']

senhasRejeitadas = ['',
                    'F1234567A',
                    'abcdefgH',
                    '1234567HI',
                    '000000',
                    '12345678',
                    'dkvb@12Q',
                    'Algo123MaisAlgo123',
                    'senhanaoaceita',
                    'CAIXAALT',
                    'TamanhO',
                    'TamanhOExtra']


print('Deverao ser aceitos:')
for i in senhasAceitas:
    print(verificaSenha(i))

print('\nDeverao ser rejeitados:')
for i in senhasRejeitadas:
    print(verificaSenha(i))
```

Resultado:

```
Deverao ser aceitos:
Senha aceita
Senha aceita
Senha aceita
Senha aceita
Senha aceita
Senha aceita
Senha aceita
Senha aceita
Senha aceita
Senha aceita
Senha aceita

Deverao ser rejeitados:
Senha rejeitada
Senha rejeitada
Senha rejeitada
Senha rejeitada
Senha rejeitada
Senha rejeitada
Senha rejeitada
Senha rejeitada
Senha rejeitada
Senha rejeitada
Senha rejeitada
Senha rejeitada
```

1.4. CPF

```
def verificaCPF(cpf):
    if re.match('[0-9]{3}[.][0-9]{3}[.][0-9]{3}[-][0-9]{2}$',cpf) ==
       None:
        return 'CPF rejeitado'
    else:
        verificacao = re.match('
           [0-9]{3}[.][0-9]{3}[.][0-9]{3}[-][0-9]{2}$',cpf).group().
           split()
        if verificacao == cpf.split():
            return 'CPF aceito'
        else:
            return 'CPF rejeitado'


cpfsAceitos = ['123.456.789-09',
               '000.000.000-00',
               '111.222.333-44',
               '789.987.123-12',
               '546.546.546-54',
               '600.070.008-09',
               '120.021.058-85',
               '879.534.654-10',
               '959.959.959-99',
               '987.987.987-98']

cpfsRejeitados = ['123.456.789-009',
                  '000.000.000.00',
                  '98798798798',
                  '123-456-789-09',
                  '000-000.000-00',
                  '987.987-987.34',
                  '123123123-12',
                  '3213.21.321-00',
                  '',
                  '123.',
                  '123.123',
                  '212.323-64',
                  '000.000.000-']


print('Deverao ser aceitos:')
for i in cpfsAceitos:
    print(verificaCPF(i))

print('\nDeverao ser rejeitados:')
for i in cpfsRejeitados:
    print(verificaCPF(i))
```

Resultado:

```
Deverao ser aceitos:
CPF aceito
CPF aceito
CPF aceito
CPF aceito
CPF aceito
CPF aceito
CPF aceito
CPF aceito
CPF aceito
CPF aceito

Deverao ser rejeitados:
CPF rejeitado
CPF rejeitado
CPF rejeitado
CPF rejeitado
CPF rejeitado
CPF rejeitado
CPF rejeitado
CPF rejeitado
CPF rejeitado
CPF rejeitado
CPF rejeitado
CPF rejeitado
```

1.5. Telefone

```
def verificaTelefone(telefone):
    if re.match('[(][0-9]{2}[)] 9[0-9]{4}[-][0-9]{4}$',telefone) ==
       None:
        return 'Telefone rejeitado'
    else:
        verificacao = re.match('[(][0-9]{2}[)] 9[0-9]{4}[-][0-9]{4}$',
           telefone).group().split()
        if verificacao == telefone.split():
            return 'Telefone aceito'
        else:
            return 'Telefone rejeitado'


telAceitos = ['(91) 99999-9999',
              '(00) 92346-7890',
              '(00) 90000-0000',
              '(02) 99999-1234',
              '(12) 91321-9874',
              '(59) 90000-2564',
              '(96) 91234-3548',
              '(98) 99999-0000',
              '(87) 91234-7777',
              '(56) 91013-7925']

telRejeitados = ['(91)99999-9999',
                 '(00) 2346-7890',
                 '(56) 01234-5678',
                 '',
                 '(0) 91234-5678',
                 '(91) 999999999',
                 '91 999999999',
                 '(00)9999-99999',
                 '(00) 00000-0000',
                 '99999999999']


print('Deverao ser aceitos:')
for i in telAceitos:
    print(verificaTelefone(i))

print('\nDeverao ser rejeitados:')
for i in telRejeitados:
    print(verificaTelefone(i))
```

Resultado:

```
Deverao ser aceitos:
Telefone aceito
Telefone aceito
Telefone aceito
Telefone aceito
Telefone aceito
Telefone aceito
Telefone aceito
Telefone aceito
Telefone aceito
Telefone aceito

Deverao ser rejeitados:
Telefone rejeitado
Telefone rejeitado
Telefone rejeitado
Telefone rejeitado
Telefone rejeitado
Telefone rejeitado
Telefone rejeitado
Telefone rejeitado
Telefone rejeitado
Telefone rejeitado
```

1.6. Data e Hora

```
def verificaDH(dh):
    if re.match('[0-9]{2}[/][0-9]{2}[/][0-9]{4}
       (00[:]00[:]00|00:00:[0-5][0-9]|00:[0-5][1-9][:][0-5][0-9]
    |00:[1-5][0-9][:][0-5][0-9]|0[1-9][:][0-5][0-9][:][0-5][0-9]|
    1[0-9][:][0-5][0-9][:][0-5][0-9]|2[0-3][:][0-5][0-9][:][0-5]
    [0-9])$',dh) == None:
        return 'Data e hora rejeitadas'
    else:
        verificacao = re.match('
           [0-9]{2}[/][0-9]{2}[/][0-9]{4}(00[:]00[:]00|
        00:00:[0-5][0-9]|00:[0-5][1-9][:][0-5][0-9]|00:[1-5][0-9]
        [:][0-5][0-9]|0[1-9][:][0-5][0-9][:][0-5][0-9]|1[0-9][:]
        [0-5][0-9][:][0-5][0-9]|2[0-3][:][0-5][0-9][:][0-5][0-9])$',dh)
           .group().split()
        if verificacao == dh.split():
            return 'Data e hora aceitas'
        else:
            return 'Data e hora rejeitadas'


dhAceitos = ['31/08/2019 20:14:55',
             '99/99/9999 23:59:59',
             '01/05/2022 19:03:57',
             '01/05/2022 00:00:00',
             '17/09/1996 05:30:25',
             '03/05/2022 19:11:00',
             '18/12/1997 13:51:42',
             '16/10/1975 02:47:22',
             '07/03/2007 12:31:19',
             '28/04/2017 01:15:00']

dhRejeitados = ['99/99/9999 3:9:9',
                '9/9/99 99:99:99',
                '99/99/999903:09:09',
                '01/05/2022 19:03',
                '28/04 01:15:00',
               '',
               '01/05/2022 50:03:57',
               '01/05/2022 00:80:00',
               '17/09/1996 05:30:90',
               '03/05/2022 19:11:99',
               '18/12/1997 13:60:42',
               '16/10/1975 02:95:22',
               '28/04/2017 01:15:0']


print('Deverao ser aceitos:')
for i in dhAceitos:
    print(verificaDH(i))

print('\nDeverao ser rejeitados:')
for i in dhRejeitados:
    print(verificaDH(i))
```

Resultado:

```
Deverao ser aceitos:
Data e hora aceitas
Data e hora aceitas
Data e hora aceitas
Data e hora aceitas
Data e hora aceitas
Data e hora aceitas
Data e hora aceitas
Data e hora aceitas
Data e hora aceitas

Deverao ser rejeitados:
Data e hora rejeitadas
Data e hora rejeitadas
Data e hora rejeitadas
Data e hora rejeitadas
Data e hora rejeitadas
Data e hora rejeitadas
Data e hora rejeitadas
Data e hora rejeitadas
Data e hora rejeitadas
Data e hora rejeitadas
Data e hora rejeitadas
Data e hora rejeitadas
```

1.7. Número real com ou sem sinal

```
def verificaNumero(numero):
    if re.match('[+-]?[0-9]+([.][0-9]+)*',numero) == None:
        return 'N mero rejeitado'
    else:
        verificacao = re.match('[+-]?[0-9]+([.][0-9]+)*',numero).group
            ().split()
        if verificacao == numero.split():
            return 'N mero aceito'
        else:
            return 'N mero rejeitado'


numerosAceitos = ['-25.467',
                  '1',
                  '-1',
                  '+1',
                  '64.2',
                  '6151.644',
                  '-6151.0',
                  '-0.000000000001',
                  '+15616651.000',
                  '+3753.6958',
                  '00.16516541']

numerosRejeitados = ['-6151.',
                     '',
                     '1.',
                     '.2',
                     '+64,2',
                     '-.3560',
                     '+.2',
                     '-.644',
                     '98-',
                     '-0,000000000001',
                     '+15616651.',]


print('Deverao ser aceitos:')
for i in numerosAceitos:
    print(verificaNumero(i))

print('\nDeverao ser rejeitados:')
for i in numerosRejeitados:
    print(verificaNumero(i))
```

Resultado:

```
Deverao ser aceitos:
N mero aceito
N mero aceito
N mero aceito
N mero aceito
N mero aceito
N mero aceito
N mero aceito
N mero aceito
N mero aceito
N mero aceito

Deverao ser rejeitados:
N mero rejeitado
N mero rejeitado
N mero rejeitado
N mero rejeitado
N mero rejeitado
N mero rejeitado
N mero rejeitado
N mero rejeitado
N mero rejeitado
N mero rejeitado
N mero rejeitado
```

2.1. Casais heterossexuais mais velhos que os filhos com pelo menos duas filhas mulheres, ou pelo menos um filho homem, ou ainda pelo menos dois filhos homens e uma filha mulher.

```
def a_2(cadeia):
    if re.match('(HM|MH)(((mm[mh]*)|([mh]*mm)|(m[mh]*m))|((h[mh]*)|([mh]*h))|((h[mh]*hm)|(hh[mh]*m)|(hhm[mh]*)|([mh]*hhm)|(m[mh]*hh)|(mh[mh]*h)|(mhh[mh]*)|([mh]*mhh)|(h[mh]*mh)|(hm[mh]*h)|(hmh[mh]*)|([mh]*hmh)))$',cadeia) == None:
        return 'Cadeia rejeitada'
    else:
        verificacao = re.match('(HM|MH)(((mm[mh]*)|([mh]*mm)|(m[mh]*m))|((h[mh]*)|([mh]*h))|((h[mh]*hm)|(hh[mh]*m)|(hhm[mh]*)|([mh]*hhm)|(m[mh]*hh)|(mh[mh]*h)|(mhh[mh]*)|([mh]*mhh)|(h[mh]*mh)|(hm[mh]*h)|(hmh[mh]*)|([mh]*hmh)))$',cadeia).group().split()
        if verificacao == cadeia.split():
            return 'Cadeia aceita'
        else:
            return 'Cadeia rejeitada'

aAceitos = ['HMmhm',
            'MHmhmhmmh',
            'MHhhhhhhhhhhhh',
            'MHh',
            'HMh',
            'MHmm',
            'HMmm',
            'HMmmmmmmmmmmm',
            'HMhmh',
            'HMmmhmhm']

aRejeitados = ['',
               'm',
               'hmh',
               'Hmhhhhh',
               'hHhhhmhhhh',
               'HmhmhmMmhmmmh',
               'mhmhmmh',
               'mHmmMhh',
               'hhhhhHM',
               'HHmmhmhm',
               'Mhmhmmhm'
               'MHm',
               'Mmm'
               'MM',]

print('Deverao ser aceitos:')
for i in aAceitos:
    print(a_2(i))

print('\nDeverao ser rejeitados:')
for i in aRejeitados:
    print(a_2(i))
```

Resultado:

```
Deverao ser aceitos:
Cadeia aceita
Cadeia aceita
Cadeia aceita
Cadeia aceita
Cadeia aceita
Cadeia aceita
Cadeia aceita
Cadeia aceita
Cadeia aceita
Cadeia aceita

Deverao ser rejeitados:
Cadeia rejeitada
Cadeia rejeitada
Cadeia rejeitada
Cadeia rejeitada
Cadeia rejeitada
Cadeia rejeitada
Cadeia rejeitada
Cadeia rejeitada
Cadeia rejeitada
Cadeia rejeitada
Cadeia rejeitada
Cadeia rejeitada
```

2.2. Casais heterossexuais mais velhos que os filhos e com uma quantidade ímpar de filhas mulheres.

```
def b_2(cadeia):
    if re.match('(HM|MH)(h*mh*|(h*mh*(h*mh*m)+)h*)$',cadeia) == None:
        return 'Cadeia rejeitada'
    else:
        verificacao = re.match('(HM|MH)(h*mh*|(h*mh*(h*mh*m)+)h*)$',
            cadeia).group().split()
        if verificacao == cadeia.split():
            return 'Cadeia aceita'
        else:
            return 'Cadeia rejeitada'


a='HM'+37*'m'
r='MH'+80*'m'

bAceitos = ['HMm',
            'MHmmm',
            'MHmhmm',
            'HMhmmm',
            'MHhmhmhmhmhmhmhm',
            'HMhhhhmhh',
            a,
            'HMhhmhhhmhhhmh',
            'HMmmmmhmmm',
            'MHhhhhmhmhhhhmmhmhhh']

bRejeitados = ['',
               'HM',
               'MHh',
                r,
               'm',
               'Hmhmmhmhm',
               'HmMh',
               'HMhhmmmhm',
               'MHmhm',
               'MHhhhhhmhhhhm']

print('Deverao ser aceitos:')
for i in bAceitos:
    print(b_2(i))

print('\nDeverao ser rejeitados:')
for i in bRejeitados:
    print(b_2(i))
```

Resultado:

```
Deverao ser aceitos:
Cadeia aceita
Cadeia aceita
Cadeia aceita
Cadeia aceita
Cadeia aceita
Cadeia aceita
Cadeia aceita
Cadeia aceita
Cadeia aceita
Cadeia aceita

Deverao ser rejeitados:
Cadeia rejeitada
Cadeia rejeitada
Cadeia rejeitada
Cadeia rejeitada
Cadeia rejeitada
Cadeia rejeitada
Cadeia rejeitada
Cadeia rejeitada
Cadeia rejeitada
Cadeia rejeitada
```

2.3. Casais heterossexuais mais velhos que os filhos, com a filha mais velha mulher e o filho mais novo homem

```
def c_2(cadeia):
    if re.match('(HM|MH)(m[hm]*h)$',cadeia) == None:
        return 'Cadeia rejeitada'
    else:
        verificacao = re.match('(HM|MH)(m[hm]*h)$',cadeia).group().
            split()
        if verificacao == cadeia.split():
            return 'Cadeia aceita'
        else:
            return 'Cadeia rejeitada'

cAceitos = ['HMmh',
            'MHmhmhmhmhmh',
            'HMmmhmmhmmmhh',
            'MHmhhhhh',
            'HMmmmmh',
            'MHmmhmmhmmh',
            'HMmmhhmhmhhhhhhmhmmh',
            'MHmmhmhmhmmmmhhhhhmmhmmh',
            'HMmhhhhhhhhhmmmh',
            'MHmmhhhmhmmmmhmmh',]

cRejeitados = ['',
               'HM',
               'MHhm',
               'MHhmmmm',
               'HMhhhh',
               'MmhH',
               'HMmmmm',
               'Mmh',
               'HMmmhmmhmhmhm',
               'Hmhmhmhmmhh']

print('Deverao ser aceitos:')
for i in cAceitos:
    print(c_2(i))

print('\nDeverao ser rejeitados:')
for i in cRejeitados:
    print(c_2(i))
```

Resultado:

```
Deverao ser aceitos:
Cadeia aceita
Cadeia aceita
Cadeia aceita
Cadeia aceita
Cadeia aceita
Cadeia aceita
Cadeia aceita
Cadeia aceita
Cadeia aceita
Cadeia aceita

Deverao ser rejeitados:
Cadeia rejeitada
Cadeia rejeitada
Cadeia rejeitada
Cadeia rejeitada
Cadeia rejeitada
Cadeia rejeitada
Cadeia rejeitada
Cadeia rejeitada
Cadeia rejeitada
Cadeia rejeitada
```

2.4. Casais homossexuais mais velhos que os filhos, com pelo menos seis filhos, em que os dois primeiros filhos formam um casal e os últimos também.

```
def d_2(cadeia):
    if re.match('(HH|MM)((mh[hm]{2,}mh)|(hm[hm]{2,}hm)|(mh[hm]{2,}hm)|(
        hm[hm]{2,}mh))$',cadeia) == None:
        return 'Cadeia rejeitada'
    else:
        verificacao = re.match('(HH|MM)((mh[hm]{2,}mh)|(hm[hm]{2,}hm)|(
            mh[hm]{2,}hm)|(hm[hm]{2,}mh))$',cadeia).group().split()
        if verificacao == cadeia.split():
            return 'Cadeia aceita'
        else:
            return 'Cadeia rejeitada'

dAceitos = ['HHmhhhhm',
            'MMhmmmmh',
            'HHhmhmhm',
            'MMmhmhhmmh',
            'HHmhmmmhmhmmhhmh',
            'MMhmhhhhhhmh',
            'MMhmmhhhmmh',
            'HHmhmmhmmhhhm',
            'HHhmmhmmhhm',
            'MMhmmhhm']

dRejeitados = ['HHmhhm',
               'MMmhhmhmmhh',
               'HHmmhhmhmmhmh',
               'MMhhhhhhhhh',
               'HHmmmmmmmm',
               'HMmhmmmhmhmmhhmh',
               'MHhmhhhhhhmh',
               'MHhmmhhhmmh',
               'HMmhmmhmmhhhm',
               'HmhmmhmmHhhhm']

print('Deverao ser aceitos:')
for i in dAceitos:
    print(d_2(i))

print('\nDeverao ser rejeitados:')
for i in dRejeitados:
    print(d_2(i))
```

Resultado:

```
Deverao ser aceitos:
Cadeia aceita
Cadeia aceita
Cadeia aceita
Cadeia aceita
Cadeia aceita
Cadeia aceita
Cadeia aceita
Cadeia aceita
Cadeia aceita
Cadeia aceita

Deverao ser rejeitados:
Cadeia rejeitada
Cadeia rejeitada
Cadeia rejeitada
Cadeia rejeitada
Cadeia rejeitada
Cadeia rejeitada
Cadeia rejeitada
Cadeia rejeitada
Cadeia rejeitada
Cadeia rejeitada
```

2.5. Casais homossexuais mais velhos que os filhos, em que o sexo dos filhos é alternado conforme a ordem de nascimento.

```
def e_2(cadeia):
    if re.match('(HH|MM)((h(?!h))|m(?!m))+$',cadeia) == None:
        return 'Cadeia rejeitada'
    else:
        verificacao = re.match('(HH|MM)((h(?!h))|m(?!m))+$',cadeia).
            group().split()
        if verificacao == cadeia.split():
            return 'Cadeia aceita'
        else:
            return 'Cadeia rejeitada'

eAceitos = ['HHh',
            'MMm',
            'HHhm',
            'HHmh',
            'MMmhm',
            'HHhmh',
            'MMmhmhm',
            'HHhmhmhm',
            'MMhmhmhmhmhmhmhmhm',
            'HHmhmhmhmhmhmhmhm']

eRejeitados = ['HMm',
               'MHh',
               'HHmm',
               'MMhh',
               'MMmmhhm',
               'HHmhmhmhmhmm',
               'MMmhhmhmhmh',
               'HHmhhhmmmhmmh',
               'MMhh',
               'HHmm']

print('Deverao ser aceitos:')
for i in eAceitos:
    print(e_2(i))

print('\nDeverao ser rejeitados:')
for i in eRejeitados:
    print(e_2(i))
```

Resultado:

```
Deverao ser aceitos:
Cadeia aceita
Cadeia aceita
Cadeia aceita
Cadeia aceita
Cadeia aceita
Cadeia aceita
Cadeia aceita
Cadeia aceita
Cadeia aceita
Cadeia aceita

Deverao ser rejeitados:
Cadeia rejeitada
Cadeia rejeitada
Cadeia rejeitada
Cadeia rejeitada
Cadeia rejeitada
Cadeia rejeitada
Cadeia rejeitada
Cadeia rejeitada
Cadeia rejeitada
Cadeia rejeitada
```

2.6. Casais homossexuais mais velhos que os filhos, com qualquer quantidade de filhos homens e mulheres, mas que não tiveram dois filhos homens consecutivos.

```
def f_2(cadeia):
    if re.match('(HH|MM)((h(?!h))|m)*$',cadeia) == None:
        return 'Cadeia rejeitada'
    else:
        verificacao = re.match('(HH|MM)((h(?!h))|m)*$',cadeia).group().
            split()
        if verificacao == cadeia.split():
            return 'Cadeia aceita'
        else:
            return 'Cadeia rejeitada'

fAceitos = ['MMmmhm',
            'HHh',
            'MMmhmh',
            'HHhmhmh',
            'MMmhmhmmmmh',
            'HHmmmmmh',
            'MMhmmmmmmh',
            'HHmmhmmhmmh',
            'MMmhmhmhmmmmmmmmhmhmmh',
            'HHmh']

fRejeitados = ['HMhh',
               'MHmhhm',
               'HHhhm',
               'MMhmhmhmhhmmm',
               'HMhhmhmhmhm',
               'MMmhmhmhmhhmm',
               'HHmhh',
               'HhmH',
               'MMmmhmhh',
               'MMmmhhmhm']

print('Deverao ser aceitos:')
for i in fAceitos:
    print(f_2(i))

print('\nDeverao ser rejeitados:')
for i in fRejeitados:
    print(f_2(i))
```

Resultado:

```
Deverao ser aceitos:
Cadeia aceita
Cadeia aceita
Cadeia aceita
Cadeia aceita
Cadeia aceita
Cadeia aceita
Cadeia aceita
Cadeia aceita
Cadeia aceita
Cadeia aceita

Deverao ser rejeitados:
Cadeia rejeitada
Cadeia rejeitada
Cadeia rejeitada
Cadeia rejeitada
Cadeia rejeitada
Cadeia rejeitada
Cadeia rejeitada
Cadeia rejeitada
Cadeia rejeitada
Cadeia rejeitada
```

2.7. Arranjo de no mínimo x $\in N$ e no máximo y $\in N$, com x > 0 , y > 0 , e x $\leq$ y , de adultos (Hs ou Ms) mais velhos que os filhos, com qualquer quantidade de filhos homens e mulheres, mas que os três filhos mais novos não foram homens.

```
def g_2(cadeia):
    if re.match('(H|M){1,7}((h|m)*m{3,}|m*)$',cadeia) == None:
        return 'Cadeia rejeitada'
    else:
        verificacao = re.match('(H|M){1,7}((h|m)*m{3,}|m*)$',cadeia).
            group().split()
        if verificacao == cadeia.split():
            return 'Cadeia aceita'
        else:
            return 'Cadeia rejeitada'

gAceitos = ['HHMmmmhmmm',
            'HMHMHMMmmhmmm',
            'MHMMHMHmhmhmmm',
            'MMMMmhhhhhmmm',
            'HHHm',
            'MHMHMMHhhhmmm',
            'MHMMHMmmm',
            'MHMHMhhhmhmmm',
            'MHMMHmhmhmhmmm',
            'MM',
            'HH',
            'HM',
            'Hmmm']

gRejeitados = ['',
               'HHMmmmhm',
               'HMHMHMMmmh',
               'MHMMHMHmhmh',
               'MMMMmhhhhh',
               'HHHh',
               'MHMHMMHhhhmm',
               'MHMMHMh',
               'MHMHMhhhmhmm',
               'MHMMHmhmhmmmmh',
               'MMmmmh',
               'HHh',
               'HhmmmM',
               'mmm',
               'Mhhm']

print('Deverao ser aceitos:')
for i in gAceitos:
    print(g_2(i))

print('\nDeverao ser rejeitados:')
for i in gRejeitados:
    print(g_2(i))
```

Resultado:

```
Deverao ser aceitos:
Cadeia aceita
Cadeia aceita
Cadeia aceita
Cadeia aceita
Cadeia aceita
Cadeia aceita
Cadeia aceita
Cadeia aceita
Cadeia aceita
Cadeia aceita
Cadeia aceita
Cadeia aceita
Cadeia aceita

Deverao ser rejeitados:
Cadeia rejeitada
Cadeia rejeitada
Cadeia rejeitada
Cadeia rejeitada
Cadeia rejeitada
Cadeia rejeitada
Cadeia rejeitada
Cadeia rejeitada
Cadeia rejeitada
Cadeia rejeitada
Cadeia rejeitada
Cadeia rejeitada
Cadeia rejeitada
Cadeia rejeitada
Cadeia rejeitada
```