Processamento de Linguagem Natural

Aula 1: *strings* e padrões

Objetivo: aplicar máquinas de estados para reconhecer padrões em *strings*.

|  |
| --- |
| **Exercício 1-** |
| *Objetivo: explicar o funcionamento de uma máquina de estados* |
| Leia atentamente o código abaixo. Ele define uma classe que deve receber, sequencialmente, um caractere a cada chamada de ler\_caractere():  class UmaMaquinaDeEstadosSimples:      def \_\_init\_\_(self):          self.estado = 0        def ler\_caractere(self, caractere):          if self.estado==0:              if caractere == 'a':                  self.estado = 1              else:                  self.estado = 0            elif self.estado == 1:              if caractere == 'a':                  self.estado = 1              else:                  self.estado = 0      def finalizar(self):          if self.estado == 1:              return 'ACEITO'          else:              return 'REJEITO'  (a) O que será impresso na tela caso executemos a seguinte chamada:  minha\_string = 'abbaa'  maquina = UmaMaquinaDeEstadosSimples()  for n in range(len(minha\_string)):      maquina.ler\_caractere(minha\_string[n])  print(maquina.finalizar())  (b) Para cada valor de n, qual é o estado que a máquina de estados assume?  (c) Encontre 3 strings que são rejeitadas pela máquina, e 3 strings que são aceitas pela máquina. |

|  |
| --- |
| **Exercício 2** |
| *Objetivo: relacionar o código de uma máquina de estados a sua representação como diagrama de bolha* |
| O diagrama abaixo representa a máquina de estados implementada no exercício 1.    (a) Como os estados são representados no código computacional?  (b) Como os estados são representados no diagrama de bolha?  (c) Como as transições entre estados são representadas no código computacional?  (d) Como as transições entre estados são representadas no diagrama de bolha?  (e) Que informação a máquina de estados usa para calcular a transição de um estado para outro?  (f) Marque no diagrama o estado inicial e o estado de “aceitação”. |

|  |
| --- |
| **Exercício 3** |
| *Objetivo: desenhar um diagrama de bolhas à partir de um código computacional* |
| Analise o código abaixo.  class ReconhecedorDeGerundismos:      def \_\_init\_\_(self):          self.estado = 0        def ler\_caractere(self, caractere):          if self.estado==0:              if caractere == 'n':                  self.estado = 1              else:                  self.estado = 0            elif self.estado==1:              if caractere == 'd':                  self.estado = 2              else:                  self.estado = 0            elif self.estado==2:              if caractere == 'o':                  self.estado = 3              else:                  self.estado = 0            elif self.estado==3:              self.estado=0      def finalizar(self):          if self.estado == 3:              return 'ACEITO'          else:              return 'REJEITO'  (a) Desenhe o diagrama de bolhas correspondente a esta máquina de estados  (b) Encontre 3 palavras existentes em português que são aceitas pela cadeia, e 3 palavras que são rejeitadas. |

|  |
| --- |
| **Exercício 4** |
| *Objetivo: construir um reconhecedor de padrões usando máquinas de estados* |
| Um problema que existe em formulários online é que, algumas vezes, o usuário digita dados inválidos nos campos. Um exemplo disso é quando há um campo de texto que pede o “ano de nascimento”, e o valor digitado é, por exemplo, o ano digitado por extenso – quando, na verdade, estamos esperando um número inteiro.  Implemente um programa que recebe uma *string* que foi digitada por um usuário hipotético e então informa se o valor digitado é um ano de nascimento válido na forma de um inteiro.  Sugestão de roteiro:  1) Faça uma lista de regras que você gostaria de implementar, incluindo as variações que você achar relevantes (por exemplo: você gostaria que fosse válido digitar tanto 1993 quanto 93? Posso ter espaços antes ou depois do número? O que acontece se eu digitar cinco dígitos? O que acontece se eu digitar anos que ainda não aconteceram, como 3022?)  2) Faça um diagrama de bolha mostrando a máquina de estados que você irá implementar  3) Implemente sua máquina de estados mapeando diretamente o diagrama de bolha para um código computacional. Se quiser, baseie-se nos códigos fornecidos como exemplo. |

|  |
| --- |
| **Exercício 5** |
| *Objetivo: relacionar expressões regulares a máquinas de estados correspondentes* |
| Existem vários tipos de strings que obedecem a padrões. Existem padrões para placas de carros (as placas antigas são compostas por três letras seguidas de quatro dígitos), para números inteiros (um ou mais dígitos seguidos), siglas de rodovias, e assim por diante. Podemos encontrar máquinas de estados para vários deles, mas é claramente pouco prático programar cadeias de *if-then-else* para cada um desses casos. Daí aparecem as expressões regulares, que são strings que permitem descrever máquinas de estados para reconhecimentos de padrões.  Numa expressão regular, um caractere qualquer corresponde exatamente a aquele caractere, na sequência que ele aparece. Isso significa que temos a correspondência:   |  |  | | --- | --- | | ab |  |   Em Python, podemos usar a biblioteca re e a função fullmatch() para verificar se uma *string* é aceita por uma expressão regular, isto é, se, ao ser inserida na máquina de estados caractere a caractere, o estado final é de aceitação:  import re  m = re.fullmatch("minha\_expressao\_regular", "minha\_string\_teste")  print(m)  A maior utilidade das expressões regulares aparece quando usamos caracteres especiais. Para cada um dos casos abaixo, (i) desenhe a máquina de estados correspondente ao exemplo de trabalho e (ii) encontre uma *string* que é aceita e uma que é rejeitada pela expressão, e teste ambas em um código Python.  (a) O caractere ‘+’ significa “um ou mais caracteres do tipo anterior a este símbolo”. Por exemplo, “a+” significa “um ou mais caracteres ‘a’”.  Exemplo de trabalho: “ab+”  (b) O caractere ‘\*’ significa “zero ou mais caracteres do tipo anterior a este símbolo”. Por exemplo, “a\*” significa “zero ou mais caracteres ‘a’”.  Exemplo de trabalho: “a\*b+”  (c) O caractere ‘?’ significa “zero ou um caractere do tipo anterior a este símbolo”. Por exemplo, “a?” significa “zero ou um caractere ‘a’”.  Exemplo de trabalho: “ab?”  (d) Quando colocamos expressões entre colchetes [ ], isso significa que trata-se de um conjunto, e a expressão aceita uma instância de qualquer um dos caracteres contidos. Por exemplo, “[abc]” significa “qualquer um entre ‘a’, ‘b’ ou ‘c’”.  Exemplo de trabalho: “a[ab]+”  (e) Se o caractere “^” é o primeiro elemento de um conjunto, então a expressão regular aceitará o complemento do conjunto. Por exemplo: “[^a]” significa “qualquer caractere, menos ‘a’.  Exemplo de trabalho: “a[^b]”  (f) O caractere ‘.’ (ponto) representa qualquer caractere. Por exemplo, “a.b” significa “’a’ seguido de qualquer caractere e então seguido de ‘b’”.  Exemplo de trabalho: “.b+”    Veja mais em: <https://docs.python.org/3/library/re.html> |

|  |
| --- |
| **Exercício 6** |
| *Objetivo: usar expressões regulares para validar campos* |
| Em expressões regulares, também temos acesso a conjuntos que são usados muito comumente:   * [0-9] significa qualquer dígito entre 0 e 9 * [a-z] significa qualquer letra minúscula, [A-Z] significa qualquer letra maiúscula * \w significa qualquer caractere que podem fazer parte de palavras, incluindo underscore (\_), números, e caracteres com acento * \W é o complemento de \w (ou: \W = [^\w] * \s significa qualquer caractere em branco ou quebra de linha * \S = [^\s] * \d significa qualquer dígito, e \D significa qualquer caractere não-digíto   Combine estas regras para gerar programas em Python que validam se uma *string* recebida como entrada representa:  (a) uma placa de carro válida, considerando o padrão antigo (3 letras seguidas de 4 dígitos).  (b) um número de telefone celular de são Paulo  (c) um CEP válido  (d) um verbo conjugado no gerúndio em português |

|  |
| --- |
| **Exercício 7** |
| *Objetivo: usar expressões regulares para validar e-mails* |
| Faça uma função em Python que recebe como entrada uma string qualquer e retorna True se essa string representa um endereço de e-mail válido, e retorna False caso contrário. |