Fatiando strings

Transcrição

Neste passo, transformaremos o algoritmo no formato de tabela em código.

Como resultado das operações, geraremos novas palavras candidatas à correção. Então começaremos definindo a função `gerador_palavras()`.

No Google Colab, esta receberá o parâmetro `palavra`. Em nosso caso, será a palavra “lgica” que estamos utilizando como exemplo.

```
def gerador_palavras(palavra):
```

COPIAR CÓDIGO

Na linha anterior, criaremos a variável `palavra_exemplo` sendo igual a `"lgica"`. A função `gerador_palavras()` retornará as `palavras_geradas` pela quarta coluna “RESULTADO” da tabela.

LGICA

ESQUERDO	DIREITO	OPERAÇÃO	RESULTADO
null	lgica	null + ó + lgica	ólgica
l	gica	l + ó + gica	**lógica**
lg	ica	lg + ó + ica	lgóica
lgi	ca	lgi + ó + ca	lgióca
lgic	a	lgic + ó + a	lgicóa
lgica	null	lgica + ó + null	lgicaó

Este resultado é originado de uma **operação**, que neste caso é a terceira coluna "OPERAÇÃO" da nossa tabela que insere uma letra. Então criaremos uma função chamada **`insere_letras()`** para fazer isso.

Dentro de `gerador_palavras()`, diremos que `palavras_geradas` é igual a `insere_letras()` recebendo as fatias da palavra, ou seja, o lado esquerdo com a primeira parte e o direito com a segunda.

Chamaremos este parâmetro de **`fatias`**. Antes de continuarmos, definiremos o corpo da função em uma nova célula com `def` recebendo as fatias.

Esta retornará um novo conjunto de palavras chamado **`novas_palavras`**. Em seguida, continuaremos a trabalhar na função `gerador_palavras()`.

A `insere_letras()` gerará a variável `palavras_geradas`, sendo justamente o que retornaremos. Porém ainda não tem as fatias, e precisaremos construí-las com a variável `fatias`, a qual será uma lista onde as passaremos para dentro da função.


```
palavra_exemplo = "lgica"
def gerador_palavras(palavra):
    fatias = []
    palavras_geradas = insere_letras(fatias)
    return palavras_geradas
```

COPIAR CÓDIGO

```
def insere_letras(fatias):

    return novas_palavras
```

COPIAR CÓDIGO

As fatias serão justamente o lado direito da primeira parte da palavra e o lado esquerdo a segunda, como já sabemos.

Então construiremos um algoritmo que fatiará a palavra. Em Python, poderemos utilizar o mesmo conceito de **Fatiamento** de Listas para `strings`.

Em uma nova célula anterior à este último bloco de código, criaremos uma variável chamada `lista` sendo igual às listas `1,2,3`.

Para acessarmos um elemento, chamaremos a `lista[]` na linha seguinte e abriremos colchetes para passarmos o `index` da posição que queremos. Neste caso, acessaremos o primeiro item da lista passando apenas o valor `0`.

```
lista = [1,2,3]
lista[0]
```

COPIAR CÓDIGO

Se quisermos o último dos três elementos, o `index` é `2`. Com isso, além de acessarmos, poderemos fatiar a lista pegando do primeiro item ao último.

Se precisarmos retornar o elemento de posição `1` até `2` com `:` entre os valores de `lista[]`, receberemos `2` como resultado, e não `3` como o esperado.

Isso aconteceu porque o sistema sempre retornará a posição anterior, ou seja, pegará até a posição `1` ao invés da `2`. Então, para retornarmos o último elemento, passaremos um número a mais, neste caso `3`.

```
lista = [1,2,3]
lista[1:3]
```

COPIAR CÓDIGO

Com isso, teremos fatiado a lista entre os elementos `2` e `3` como queríamos.

Faremos algo similar com a `string`. Passaremos `"lgica"` para a variável `lista` ao invés de `[1,2,3]`, e pegaremos as posições `1` relativa ao "l" até a `2` que é o "i" dentro de `lista[]`.

```
lista = "lgica"
lista[1:2]
```

COPIAR CÓDIGO

Como retorno, receberemos a parte `'gi'` da palavra.

Portanto, trabalhamos com Fatiamento de `strings` da mesma maneira feita com listas, o que nos ajudará bastante.

Para entendermos melhor como fatiaremos as palavras, precisaremos pegar o lado direito e esquerdo por meio da criação de uma **tupla**, colocando a `lista[]` entre parênteses.

O primeiro elemento da tupla será representado pela primeira parte do lado esquerdo e o segundo pela segunda fatia.

```
lista = "lgica"
(lista[:1],lista[1:])
```

COPIAR CÓDIGO

O retorno com o lado esquerdo da primeira posição é o `'l'` e do lado direito é o `'gica'`. Em seguida, poderemos apenas variar os números das partes da tupla.

De volta ao `gerador_palavras()`, construiremos a função com um laço `for` onde `i` será as posições de dentro da palavra com **`range()`** recebendo `len()` da `palavra`.

Em seguida, adicionaremos a operação da tupla às `fatias` da lista com `append()`, sem nos esquecermos dos parênteses para criar a tupla dentro.

Ao invés de passarmos a posição `1` no fatiamento da `string`, passaremos o `i`.

Desta forma, adicionaremos as tuplas à lista de fatias, representando o lado direito e o esquerdo da palavra.

Comentaremos as duas últimas linhas do bloco de `gerador_palavras()` para testarmos o código sem termos erros, afinal não retornarão nada.

Em seguida, imprimiremos as fatias da variável `fatias` e chamaremos a função `gerador_palavras()` recebendo a `palavra_exemplo`.

```
palavra_exemplo = "lgica"
def gerador_palavras(palavra):
    fatias = []
    for i in range(len(palavra)):
        fatias.append((lista[:i],lista[i:]))
    print(fatias)
    #palavras_geradas = insere_letras(fatias)
    #return palavras_geradas

gerador_palavras(palavra_exemplo)
```

COPIAR CÓDIGO

Como retorno, teremos justamente a simulação dos resultados das operações com cinco itens separados entre lados esquerdos e direitos.

Para fins de comparação com os resultados da última coluna da tabela do algoritmo, veremos que o primeiro elemento possui a primeira tupla `null` ou vazia como `''`, e `'lgica'` como segunda. Em

seguida, teremos apenas a letra `'l'` como lado esquerdo e o `'gica'` como direito, até o último caso.

Porém, o último caso retornado possui `'lgic'` do lado esquerdo, e apenas `'a'` do direito, diferente do `'lgica'` seguido de `null` que está presente na última linha da coluna "RESULTADO". Também recebemos cinco elementos como retorno do código, e não seis como é a quantidade de palavras geradas na tabela, então um resultado está faltando.

No exemplo de Fatiamento que fizemos no início, vimos que é preciso adicionar um número a mais para pegarmos o último caractere da `string`. Da mesma forma, adicionaremos `+1` ao `len()` da `palavra` dentro de `range()`.

```
palavra_exemplo = "lgica"
def gerador_palavras(palavra):
    fatias = []
    for i in range(len(palavra)+1):
        fatias.append((lista[:i],lista[i:]))
    print(fatias)
    #palavras_geradas = insere_letras(fatias)
    #return palavras_geradas

gerador_palavras(palavra_exemplo)
```

COPIAR CÓDIGO

Com isso, receberemos os mesmos seis resultados de palavras fatiadas que podem ser vistos na tabela, sendo o último `('lgica','')` como esperado.

A seguir, criaremos o algoritmo da operação e o definiremos em `insere_letras()`.