

ASIMOV

Strings

As strings são usadas em Python para registrar informações de texto, como nome. As cordas em Python são na verdade uma *seqüência*, o que basicamente significa que o Python acompanha cada elemento da seqüência de caracteres como uma seqüência. Por exemplo, Python entende a string "hello" como uma seqüência de letras em uma ordem específica. Isso significa que poderemos usar a indexação para pegar letras particulares (como a primeira letra ou a última letra).

Essa idéia de uma seqüência é importante em Python e nós vamos abordá-la mais tarde.

Nesta palestra, aprenderemos os seguintes tópicos:

1. Criando Strings
2. Impressão de strings
3. Indexação e corte de strings
4. Propriedades da Cadeia de Caracteres
5. Métodos de Strings
6. Formatação de impressão

Criando uma String

Para criar uma string em Python, você precisa usar aspas simples ou aspas duplas. Por exemplo:

```
In [1]: # Uma palavra  
        'hello'
```

```
Out[1]: 'hello'
```

```
In [2]: # Uma frase inteira  
        'This is also a string'
```

```
Out[2]: 'This is also a string'
```

```
In [3]: # Também é possível usar aspas duplas  
        "String built with double quotes"
```

```
Out[3]: 'String built with double quotes'
```

Imprimindo uma String

Usando o Jupyter Notebook com apenas uma sequência de caracteres em uma célula emitirá automaticamente cadeias de caracteres, mas a maneira correta de exibir cadeias na sua saída é usando uma função de impressão.

```
In [4]: # Podemos simplesmente declarar uma string  
        'Hello World'
```

```
Out[4]: 'Hello World'
```

```
In [5]: # Note que podemos imprimir várias strings assim  
        'Hello World 1'  
        'Hello World 2'
```

```
Out[5]: 'Hello World 2'
```

Mas a maneira correta (inclusive para outras IDEs) é utilizar o método print().

```
In [6]: print('Hello World 1')  
        print('Hello World 2')  
        print('Use \n to print a new line')  
        print('\n')  
        print('See what I mean?')
```

```
Hello World 1  
Hello World 2  
Use  
  to print a new line
```

```
See what I mean?
```

Nós também podemos usar uma função chamada len() para verificar o comprimento de uma string!

```
In [33]: len('Hello World')
```

```
Out[33]: 11
```

Indexação em Strings

Sabemos que as strings são uma sequência, o que significa que o Python pode usar índices para chamar partes da sequência. Vamos aprender como isso funciona.

Em Python, usamos colchetes [] após um objeto para chamar seu índice. Devemos também notar que a indexação começa em 0 para Python. Vamos criar um novo objeto chamado "s" e a caminharos através de alguns exemplos de indexação.

```
In [2]: # Define s como uma string  
        s = 'Hello World'
```

```
In [8]: # Checa  
        s
```

```
Out[8]: 'Hello World'
```

```
In [9]: # Printa o objeto  
print(s)
```

Hello World

Vamos começar a indexar!

```
In [10]: # Mostra o primeiro elemento (neste caso uma letra)  
s[0]
```

Out[10]: 'H'

```
In [22]: s[1]
```

Out[22]: 'e'

```
In [23]: s[2]
```

Out[23]: 'l'

Podemos usar um `:` para executar corte que pega tudo até um ponto designado. Por exemplo:

```
In [11]: # Retorna todos elementos a partir do elemento de indice 1  
s[1:]
```

Out[11]: 'ello World'

```
In [12]: # Observe que não há mudanças no elemento s  
s
```

Out[12]: 'Hello World'

```
In [14]: # Retorna tudo até o elemento de índice 3  
s[:3]
```

Out[14]: 'Hel'

Observe o corte acima. Aqui, estamos dizendo ao Python que pegue tudo de 0 a 3. Não inclui o 3º índice. Você notará muito isso em Python, onde as declarações e geralmente são no contexto "até, mas não incluindo".

```
In [15]: # Tudo  
s[:]
```

Out[15]: 'Hello World'

Também podemos usar indexação negativa para retroceder.

```
In [16]: # Última letra (um índice antes do 0, então ele começa da parte de trás)  
s[-1]
```

Out[16]: 'd'

```
In [18]: # Pega tudo, menos a última letra
s[:-1]
```

Out[18]: 'Hello Worl'

Também podemos usar notação de índice e fatia para capturar elementos de uma sequência com espaçamentos (o espaçamento padrão é 1). Por exemplo, podemos usar dois dois pontos em uma linha e, em seguida, um número que especifica a frequência para capturar elementos. Por exemplo:

```
In [42]: # Pega tudo, de 1 em 1
s[::1]
```

Out[42]: 'Hello World'

```
In [19]: # Pega tudo, mas os espaçamentos são de 2 em 2
s[::2]
```

Out[19]: 'HloWrd'

```
In [21]: # Pega tudo, mas com passos negativos, de trás para frente.
s[::-1]
```

Out[21]: 'dlrow olleH'

Propriedades das Strings

É importante notar que as strings têm uma propriedade importante conhecida como imutabilidade. Isso significa que, uma vez que uma string é criada, os elementos nele não podem ser alterados ou substituídos. Por exemplo:

```
In [48]: s
```

Out[48]: 'Hello World'

```
In [3]: # Vamos tentar mudar a primeira letra para 'x'
s[0] = 'x'
```

```
-----
TypeError                                Traceback (most recent call last)
<ipython-input-3-bf1b55f1477a> in <module>()
      1 # Vamos tentar mudar a primeira letra para 'x'
----> 2 s[0] = 'x'
```

TypeError: 'str' object does not support item assignment

Observe como o erro nos diz diretamente o que não podemos fazer, alterar a atribuição do item!

Algo que podemos fazer é concatenar strings!

```
In [50]: s
```

```
Out[50]: 'Hello World'
```

```
In [4]: # Concatenar as strings  
s + ' concatenate me!'
```

```
Out[4]: 'Hello World concatenate me!'
```

```
In [5]: # Assim podemos redefinir completamente s  
s = s + ' concatenate me!'
```

```
In [6]: print(s)
```

```
Hello World concatenate me!
```

```
In [7]: s
```

```
Out[7]: 'Hello World concatenate me!'
```

Podemos usar o símbolo de multiplicação para criar repetições!

```
In [8]: letter = 'z'
```

```
In [9]: letter*10
```

```
Out[9]: 'zzzzzzzzzz'
```

Métodos embutidos em strings

Os objetos em Python geralmente possuem métodos internos. Esses métodos são funções dentro do objeto (aprenderemos sobre isso em muito mais profundidade depois) que podem executar ações ou comandos no próprio objeto.

Chamamos métodos com um ponto e depois o nome do método. Os métodos estão na forma: objeto.método(parâmetros)

Onde os parâmetros são argumentos extras que podemos passar para o método. Não se preocupe se os detalhes não fazem 100% de sentido agora. Mais tarde, criaremos nossos próprios objetos e funções!

Aqui estão alguns exemplos de métodos internos em strings:

```
In [10]: s
```

```
Out[10]: 'Hello World concatenate me!'
```

```
In [11]: # Coloca toda string em caixa alta  
s.upper()
```

```
Out[11]: 'HELLO WORLD CONCATENATE ME!'
```

```
In [12]: # Caixa baixa  
s.lower()
```

```
Out[12]: 'hello world concatenate me!'
```

```
In [13]: # Divide uma string nos espaços em branco (este é o padrão)  
s.split()
```

```
Out[13]: ['Hello', 'World', 'concatenate', 'me!']
```

```
In [14]: # Divide em um elemento específico (não inclui o elemento que foi dividido)  
s.split('W')
```

```
Out[14]: ['Hello ', 'orld concatenate me!']
```

Existem outros métodos do que os abrangidos aqui. Visite a seção de String avançada para descobrir mais!

Formatação de impressão

Podemos usar o método `.format()` para adicionar objetos formatados a instruções de impressões.

A maneira mais fácil de mostrar isso é através de um exemplo:

```
In [15]: 'Insert another string with curly brackets: {}'.format('The inserted string')
```

```
Out[15]: 'Insert another string with curly brackets: The inserted string'
```