Strings

Já vimos um pouco sobre strings no capítulo 2, sobre tipos de dados que o Python utiliza. Resgatando a definição que usamos, são dados definidos entre aspas duplas (“) ou simples (‘), representando texto, que normalmente é exibido por um programa ou exportado por ele.

Resumindo e simplificando, basicamente, string é um texto, podendo conter letras, números, símbolos.

Definindo Strings

Apenas para deixar o capítulo completo, vamos relembrar como definimos uma string. Uma string é definida como tudo que se encontra entre aspas simples (‘) ou duplas (“). Qual você usa é uma questão de gosto e opinião pessoal. Desta forma vamos aos primeiros exemplos:

```
string1 = "Oi Python"
string2 = 'Tchau Python'
print(string1)
print(string2)

> Oi Python
> Tchau Python
```

Desta forma, vimos acima as duas principais formas de definir uma string. Existe uma terceira forma, que é a definição de strings entre três aspas triplas. Neste caso, caracteres escapados (veremos o que são em alguns instantes) assumem seu significado correto, e quebras de linha, seja no terminal ou na linha de código também são interpretadas como tal no output. Veja o exemplo abaixo, onde cada quebra de linha foi feita dando ENTER após cada sentença. A string só se encerra ao fechá-la com o mesmo conjunto de aspas simples ou duplas utilizado para iniciá-la:

```
string_grande = '''Olá. Esta é uma string grande no Python.
Aqui você pode usar " ou '
Caracteres são escapados como se espera.
É a terceira forma de definir uma string, apesar de não ser tão usual....
\t testando o TAB para finalizar'''

> Olá. Esta é uma string grande no Python.
> Aqui você pode usar " ou '
> Caracteres são escapados como se espera.
> É a terceira forma de definir uma string, apesar de não ser tão usual....
```

```
>       testando o TAB para finalizar
```

Escapando strings

Ok, tá bom. Mas e se eu quiser criar uma string com aspas duplas e simples no meio dela? Bem, se você usa aspas duplas em sua string e quiser usar aspas simples nela, ou vice-versa, não há qualquer problema. Basta colocá-los junto com a string e ela continuará sendo válida. Veja:

```
string3 = "A cantora Sinnead O'Connor"
string4 = 'Alfredo disse "Corram aqui para ver isso!"'
print(string3)
print(string4)

> A cantora Sinnead O'Connor
> Alfredo disse "Corram aqui para ver isso!"
```

Agora, caso você queira utilizar as duas em uma string, ou caso sempre defina suas strings com aspas duplas e queira usar aspas duplas em uma string específica, temos que escapar este caracter. Escapar? Sim, escapar. Escapar significa inserir em uma string caracteres que tenham algum significado especial, como uma nova linha, um caracter de tabulação (TAB) ou as aspas simples ou duplas.

Para escapar um caracter em Python utilizamos a contrabarra (\) antes do elemento que desejamos escapar. Vejamos os exemplos:

```
string5 = "Alfredo disse \"Corram aqui para ver isso!\""
string6 = 'Sinnead O\'Connor disse "Nothing compares 2 u"'
print(string5)
print(string6)

> Alfredo disse "Corram aqui para ver isso!"
> Sinnead O'Connor disse "Nothing compares 2 u"
```

E se eu quiser usar uma contrabarra na minha string? Fácil, basta escapar ela também. Ao definir uma string, duas contrabarras (\\) viram uma contrabarra na saída:

```
string7 = "Estou escapando uma \\"
print(string7)

> Estou escapando uma \
```

Retirado da documentação e traduzido por mim mesmo, segue abaixo uma tabelinha com sequências escapadas e seus significados:

Sequência Escapada	Significado
\\	Contrabarra
\’	Aspas simples
\”	Aspas duplas
\a	Caracter de controle ASCII Bell (BEL)
\b	Caracter ASCII Backspace (BS)
\f	Caracter ASCII Formfeed (FF)
\n	Caracter ASCII Linefeed (LF) - este cria uma nova linha no output
\r	Caracter ASCII Carriage Return (CR)
\t	Caracter ASCII Tab Horizontal (TAB)
\v	Caracter ASCII Tab Vertical (VT)
\ooo	Caracter ASCII com valor octal “ooo”
\xhh	Caracter ASCII com valor hex de “hh”

Strings como elas são

Uma outra forma de representar strings é antecedendo sua definição com a letra “r”. Isto gera o que chamamos de raw string literals, onde a string é exibida exatamente como ela foi definida, e não substituindo caracteres escapados por seus significados no output. Vejamos o exemplo abaixo para entender:

```
string8 = r"Utilizamos \n para inserir uma nova linha na string"
print(string8)

> Utilizamos \n para inserir uma nova linha na string
```

Desta forma, podemos escrever strings sem se preocupar com caracteres escapados ou qualquer coisa do tipo.

Inserindo variáveis em uma string

Outra situação bastante recorrente: tenho um determinado valor armazenado em uma variável e quero exibi-lo juntamente com outros caracteres de texto. Posso

fazer isso de diferentes formas. A primeira delas é utilizando o sinal de "%", que é um caractere de formatação de strings no Python.

Apesar de permitir utilizações mais complexas para formatação de strings, sua utilização mais básica é através do %s para incluir outras strings em uma string, e %d para incluir um número inteiro (Integer). Vejamos exemplos dos dois:

```
nome = "Felipe"
idade = 30
print("Meu nome é %s " % nome)
print("Tenho %d anos" % idade)

> Meu nome é Felipe
> Tenho 30 anos
```

Também podemos incluir números decimais (float) em strings, usando %f. Vejamos algumas possibilidades no exemplo abaixo. Primeiro, um exemplo sem qualquer formatação:

```
a = 30.46257
print("Formatando decimais: %f" % a)

> Formatando decimais: 30.462570
```

Se colocamos um ponto e a quantidade de casas decimais, definimos a quantidade de algarismos a serem impressos após o ponto:

```
print("Formatando decimais: %.2f" % a)
print("Formatando decimais: %.3f" % a)

> Formatando decimais: 30.46
> Formatando decimais: 30.463
```

Você pode inserir mais de um valor na string, colocando-os em ordem dentro de parênteses, separados por vírgulas. Esta estrutura se chama Tupla, e em um capítulo posterior, a veremos em detalhes:

```
nome = 'Felipe'
idade = 30
print("Meu nome é %s e tenho %d anos" % (nome, idade))

Meu nome é Felipe e tenho 30 anos
```

Concatenação

Outra forma de inserir valores em strings é através da concatenação. O sinal de concatenação no Python é o "+". Através dele, juntamos strings com variáveis e outras strings. Vejamos no exemplo abaixo:

```
nome = 'Felipe'
print("Olá, meu nome é " + nome)

> Olá, meu nome é Felipe
```

Esta forma tem um porém. Vejamos agora o que acontece se incluirmos a variável idade, um número inteiro, nesta string, por concatenação:

```
nome = 'Felipe'
idade = 30
print("Olá, meu nome é " + nome + " e tenho " + idade + " anos.")

> Traceback (most recent call last):
>   File "<stdin>", line 1, in <module>
> TypeError: Can't convert 'int' object to str implicitly
```

A função print não faz a conversão de um número inteiro para uma string automaticamente. Desta forma, para incluir esta variável, precisamos converter o número inteiro para uma string usando a função str, e passando dentro do parênteses o valor a ser convertido. Veja:

```
print("Olá, meu nome é " + nome + " e tenho " + str(idade) + " anos.")

> Olá, meu nome é Felipe e tenho 30 anos.
```

E no caso de números decimais? Bem, se você não quer formatá-lo, basta chamar a função str da mesma forma que no exemplo anterior:

```
valor = 503.78987
print("O valor é " + str(valor))

> O valor é 503.78987
```

Mas, se quiser formatá-la antes, teremos que usar outra função, a função format. Seus parâmetros são o valor a ser formatado e o formato das casas decimais, tal qual vimos na formatação de decimais nos exemplos anteriores, onde passamos o ponto e o número de casas decimais após o mesmo. Vejamos no exemplo:

```
print("O valor é " + format(valor, '.2f'))


> O valor é 503.79
```

Note que a função format já retorna uma string, não havendo aqui a necessidade de converter o resultado em string usando a função str.

Strings como listas

Em Python, toda string é tratada como uma lista. Ainda falaremos mais de listas em capítulos posteriores, mas resumindo, uma lista é uma estrutura semelhante aos arrays, para quem vem de outras linguagens de programação. São um conjunto de dados armazenados em uma única estrutura. E em Python, uma string é uma lista, ou uma sequência de caracteres.

Uma das situações que isso desencadeia é que podemos acessar qualquer caractere da string através de seu índice, ou index, que é a sua posição na string. A primeira letra tem índice 0 (zero), e a última é igual a quantidade de caracteres da string menos um. Vejamos:

```
string9 = "Olá, meu nome é Felipe"
print(string9[0])
print(string9[3])
print(string9[21])


> O
> ,
> e
```

No exemplo acima, a string que usamos tem 22 caracteres. E o que acontece se usarmos um índice que não existe? O Python retorna um erro. Vejamos:

```
print(string9[25])


> Traceback (most recent call last):
>   File "<stdin>", line 1, in <module>
> IndexError: string index out of range
```

No capítulo onde falaremos sobre listas veremos mais sobre como trabalhar com elas de forma mais eficiente.

Funções úteis para Strings

Aqui, vamos apresentar algumas funções úteis para strings, explicando um pouco o funcionamento de cada uma delas. Não vou entrar em detalhes de todas as funções existentes no Python, pois são muitas e a documentação da linguagem é até melhor que eu para explicar as mesmas. Vou apenas apresentar algumas das que considero importantes e mais utilizadas e explicá-las com bons exemplos.

Chamamos as funções abaixo, como veremos nos exemplos, como métodos da própria string.

capitalize(), lower() e upper()

A função "*capitalize()*" transforma a primeira letra de uma string em maiúscula. "*lower()*" e "*upper()*" transformam, respectivamente, todas as letras em minúsculas e maiúsculas. Veja alguns exemplos:

```
string10 = "olá, meu NOME é Felipe"
print(string10.capitalize())
print(string10.upper())

> Olá, meu nome é felipe
> OLÁ, MEU NOME É FELIPE
```

center(), ljust() e rjust()

A função "center()", como o nome provavelmente já dá a entender, centraliza uma string para um determinado número de caracteres, utilizando um caractere a ser definido para preencher a string dos dois lados. "ljust()" e "rjust()" fazem o mesmo, mas preenchem caracteres apenas à esquerda ou à direita. Veja um exemplo com o center. Os outros dois são análogos:

```
string11 = "olá, meu nome é Felipe"
print(string11.center(50,"*"))

> **********olá, meu nome é Felipe**********
```

find()

Este método indica se uma determinada substring faz parte de uma string. Caso faça, ela retorna a posição onde começa esta substring. Caso não, a função retorna -1. Veja no exemplo:

```
string12 = "Olá, meu nome é Felipe"
substring1 = "meu"
print(string12.find(substring1))
substring2 = "José"
```

```
print(string12.find(substring2))

> 5
> -1
```

Você também pode definir os índices a partir dos quais a busca inicia e termina, através dos parâmetros beg (primeiro após a string a ser encontrada) e end (segundo após a string, logo após o parâmetro beg), conforme o exemplo abaixo:

```
print(string12.find(substring1, 7))
print(string12.find(substring1, 2))

> -1
> 5
```

isalnum(), isalpha() e isnumeric()

Estas funções indicam se uma determinada string é, respectivamente, totalmente formada por caracteres alfanuméricos, alfabéticos (só contem letras) e números. Se as strings tiverem pelo menos um caracter que invalide cada condição, estas funções retornarão “False”. Vale notar que espaço em branco na string torna uma string inválida para qualquer uma destas funções. Vamos com muitos exemplos, para podermos entender bem. Primeiro, uma string só com letras:

```
string13 = "Felipe"
print(string13.isalnum())
print(string13.isalpha())
print(string13.isnumeric())

> True
> True
> False
```

Agora, uma string só com números:

```
string14 = "1234"
print(string14.isalnum())
print(string14.isalpha())
print(string14.isnumeric())

> True
> False
> True
```

E uma com os letras e números:

```
string15 = "Felipe1234"
print(string15.isalnum())
```

```
print(string15.isalpha())
print(string15.isnumeric())

> True
> False
> False
```

len()

A função *"len()"* retorna a quantidade de caracteres presentes em uma determinada string. Repare que a função "*len()*", diferentemente da maioria das funções que vimos, não é chamada através de uma string, mas a string é passada como um argumento para ela. Veja:

```
string16 = "Meu nome é Felipe"
print(len(string16))

> 17
```

replace()

A função "*replace()*", como o nome já indica, substitui uma parte de uma string por outra parte, definidas nos dois primeiros argumentos da função. Veja o exemplo:

```
string17 = "Olá, meu nome é Felipe"
print(string17.replace("Felipe","José"))

> Olá, meu nome é José
```

strip(), rstrip() e lstrip()

A função "*strip()*" remove espaços no começo e no fim das strings. Já as funções "*rstrip()*" e "*lstrip()*" removem, respectivamente, espaços à direita e à esquerda da string (r e l de right e left). Vejamos alguns exemplos:

```
string18 = "   Olá, meu nome é Felipe      "
print(string18)
print(string18.strip())
print(string18.rstrip())
print(string18.lstrip())

>    Olá, meu nome é Felipe
> Olá, meu nome é Felipe
>   Olá, meu nome é Felipe
```

```
> Olá, meu nome é Felipe
```

PS: Não dá para conferir muito bem o resultado da função “rstrip()”, mas eu juro que ela funciona.

Conclusão

Existem muitas outras funções úteis relacionadas a strings no Python, mas checamos algumas das mais importantes e mais utilizadas. Vimos também como definir strings, incluir o valor de variáveis nelas e formatar números a serem inseridos. Assim como o tratamento que o Python dá às strings, tratando-as como listas.