Strings

1. Strings
No primeiro capítulo mencionamos quatro tipos primitivos de dados: inteiro, real, lógico e texto/literal (string). Na verdade, o quarto tipo básico seria um caractere. Uma string é uma coleção de caracteres - como se fosse uma lista, mas aceitando apenas elementos textuais. Vamos verificar algumas propriedades das strings!

1.1. Acessando elementos em uma string
No capítulo sobre Listas, vimos que podemos acessar elementos individuais de uma lista utilizando um índice entre colchetes. Vamos testar a mesma lógica com strings?

```
frase = "Let's Code"
print(frase[0])
print(frase[1])
print(frase[2])
print(frase[3])
print(frase[4])
```
Note que o programa acima imprime "Let's", com um caractere por linha. Ou seja, ele considerou frase[0] como "L", frase[1] como "e", e assim sucessivamente. Uma string é, de fato, uma coleção de caracteres.

Porém, ao contrário de uma lista, dizemos que uma string é imutável. Isso significa que não podemos alterar um elemento individual da string. O programa abaixo produz um erro:

```
frase = "let's code"
frase[0] = 'L'
```
Para alterar uma string, é necessário redefini-la, de modo que a string original será descartada e a nova (alterada) será escrita por cima da original. Ou, alternativamente, podemos gerar uma cópia da string com alterações. Veremos mais detalhes adiante.

1.2. Operações entre strings
1.2.1. Operadores aritméticos
Alguns operadores aritméticos funcionam com strings também. Naturalmente, eles não servem para fazer contas, mas nos permitem fazer de forma intuitiva algumas operações bastante úteis.

O operador + serve como um operador de concatenação de strings: unir duas strings. Observe o exemplo abaixo:

```
string1 = 'Olá'
string2 = 'Mundo'
resultado = string1 + string2
print(resultado) # Na tela: 'OláMundo'
```
Outro operador que funciona é o operador *. Este operador não é usado entre duas strings, mas entre uma string e um int. Ele repetirá a string o número de vezes dado pelo int.

```
string = 'Olá mundo!'
```

```
multi = string * 3
print(multi) # Na tela: 'Olá mundo!Olá mundo!Olá mundo!'
```

1.2.2. Operadores lógicos

Os operadores lógicos (<, >, <=, =>, != e ==) também funcionam com strings. Esses operadores são case sensitive, ou seja, diferenciam maiúsculas de minúsculas.

De maneira muito simplificada e desconsiderando diferenças entre maiúsculas e minúsculas, podemos dizer que eles consideram ordem alfabética: 'banana' é maior do que 'abacaxi'.

A explicação mais completa é a seguinte: internamente, cada caractere é armazenado como um número. Quando utilizamos qualquer tipo de aplicação que irá exibir um texto (um editor de textos, navegador de internet, ou mesmo os nossos programinhas em Python rodando no terminal), a aplicação usa esses números como índices em uma tabela de codificação de caracteres.

Temos diversos esquemas diferentes de codificação de caracteres em uso pelo mundo, e quando você está usando um programa ou navegando por um site e você nota símbolos estranhos no texto (frequentemente onde teríamos caracteres especiais, como letras com acento), é provável que o autor do texto tenha utilizado uma tabela e o seu computador esteja usando outra.

Vários programas permitem a conversão entre essas tabelas, e você já deve ter visto essa "sopa de letrinhas" em alguma aba ou janela de configuração em algum editor de textos: utf-8, utf-16, windows-1252 (ou cp-2152), e até mesmo alguns padrões ISO.

Para ilustrar a ideia, vamos colocar aqui uma das tabelas mais simples, a tabela ASCII:

Fonte:
https://upload.wikimedia.org/wikipedia/commons/thumb/1/1b/ASCII-Table-wide.svg/1200px-ASCII-Table-wide.svg.png

Note que o caractere 'A' está no índice 65, o 'B' está no índice 66, e assim sucessivamente. Por isso, 'Abacate' < 'Banana' é verdadeiro: a primeira string começa com uma letra no índice 65 da tabela, a segunda com uma letra no índice 66.

No caso de 'Abacate' e 'Abacaxi', temos um "empate" das 5 primeiras letras. Então é a sexta letra que vai mandar: 'x' é maior do que 't', por estar em uma posição superior na tabela.

Note que a ordem não é exatamente alfabética: entre os caracteres maiúsculos, seguimos ordem alfabética. Entre os minúsculos, idem. E entre os dígitos numéricos, também temos ordem crescente correspondente aos valores. Mas todos os minúsculos são "maiores" do que qualquer maiúsculo, que por sua vez são "maiores" do que qualquer dígito numérico. Símbolos, operadores e sinais de pontuação estão em posições diversas.

1.2.3. Copiando uma string através de concatenação

Caso você já tenha resolvido problemas de somatório (a essa altura, espera-se que tenha resolvido vários!), você já deve estar acostumado a utilizar um loop onde novos valores são somados em uma mesma variável. Somar os números de uma lista, por exemplo, tem mais ou menos essa carinha:

```
soma = 0
for numero in lista:
    soma = soma + numero
```

A mesma lógica pode ser aplicada a uma string:

```
string_inicial = 'Olá Mundo'
string_final = '' # cria uma string vazia
for letra in string_inicial:
    string_final = string_final + letra
print(string_final)
```

Isso é útil porque antes de "somar" cada letra à string final podemos fazer alterações (como transformar em maiúscula ou minúscula, acrescentar caracteres entre 2 letras etc.). É um jeito de fazer tratamento de strings. Veremos mais sobre tratamento de strings no capítulo de funções de strings.

1.3. Transformando uma string em lista

Strings são imutáveis, e isso pode nos dar um pouco de trabalho quando queremos fazer pequenas alterações, como forçar um caractere a ser maiúsculo ou acrescentar um caractere à string. Uma das formas de fazer envolve a "soma cumulativa" apresentada acima. Outra forma envolve transformar a nossa string em lista, que é uma estrutura mutável. Execute o programa abaixo:

```
string = "let's Code"
lista = list(string)
lista[0] = 'L'
lista.append('!')
print(lista)
# resultado:
# ['L', 'e', 't', "'", 's', ' ', 'C', 'o', 'd', 'e', '!']
```

Como a lista é mutável, nela conseguimos alterar uma letra e adicionar um símbolo ao final sem dificuldades! Porém, infelizmente nosso resultado é uma lista, o que não ficou muito legível para o usuário. Podemos resolver isso utilizando a função join. Veremos em breve como ele realmente funciona, mas por hora podemos utilizá-lo da seguinte maneira para transformar lista em string:

```
string_original = "let's Code"
lista = list(string_original)
lista[0] = 'L'
lista.append('!')
string_final = ''.join(lista) # antes do . temos uma string vazia
print(string_final)
# resultado: "Let's Code!"
```

Para as modificações mais comuns, temos algumas funções prontas que poderão ser bastante úteis!

2. Símbolos especiais

Além de letras, números, sinais de pontuação, símbolos matemáticos etc., uma string pode conter alguns operadores especiais de controle. Esses operadores podem indicar, por exemplo, uma quebra de linha ou uma tabulação. Vejamos os mais comuns:

2.1. Quebra de linha

Uma quebra de linha indica que o programa exibindo a string deverá quebrar a linha atual e exibir o restante da string na linha seguinte, e é representada na maioria dos sistemas e na web pelo símbolo \n. Execute o programa abaixo e veja o resultado na tela:

```
print('Olá\nMundo')
```

2.2. Tabulação

A tabulação indica um recuo equivalente ao da tecla Tab - um recuo de início de parágrafo, ou o recuo que usamos para aninhar linhas de código em Python. Ela é representada pelo símbolo \t. Verifique o resultado do exemplo abaixo:

```
aprovados = ['Mario', 'Peach', 'Luigi']
reprovados = ['Wario', 'Bowser']

print('Candidatos aprovados:')
for nome in aprovados:
    print('\t', nome)

print('Candidatos reprovados:')
for nome in reprovados:
    print('\t', nome)
```

2.3. Barra

E se nós quiséssemos representar uma string que explica o significado de \n, por exemplo, como proceder? Afinal, ao ver o símbolo \n o programa entenderá que é uma quebra de linha e fará isso ao invés de escrever \n na tela.

Podemos utilizar 2 barras: \\. Ao fazermos isso, o programa entende que é para representar a barra na tela ao invés de interpretá-la como início de outro símbolo especial.

```
print('Utilizamos o \\n para quebrar linhas.')
```

2.4 Aspas

Um problema que você deve ter se deparado é que parece impossível representar o símbolo ' em uma string que foi aberta por esse símbolo, já que a segunda ocorrência dele fechará a string. Idem para o símbolo ". Podemos resolver isso da mesma forma que fizemos com a barra: \' irá sempre representar o símbolo ' e \" irá sempre representar o símbolo " ao invés de fechar uma string.

```
print('Imprimindo uma aspa simples(\') dentro de uma string sem problemas')
print("Imprimindo aspas duplas(\") dentro de uma string sem problemas")
```