Introdução à Programação

Aula 12: Tuplas, Strings e Conceitos Importantes do Python

Prof. Eduardo Corrêa Gonçalves

14/05/2024

Sumário

Tupla (1/4)

- **Tupla (*tuple*) - Definição**
  - Estrutura idêntica à lista (coleção ordenada de elementos), porém **imutável**.
    - Isso significa que:
      - Você **não** pode adicionar, remover ou alterar elementos.
      - Mas pode recuperar elementos (pois essa operação não altera a tupla).

  - Importante para:
    - Definir **constantes**.
    - Trabalhar com **dados protegidos**.
    - Trabalhar com a técnica de **atribuição múltipla**.

Tupla (2/4)

- **Criando Tuplas**
  - Devemos especificar uma sequência de valores entre parênteses, onde os valores devem estar separados por vírgula:

```
t1 = (10, 20, 30, 40, 50)

t2 = (100,)              # tupla com um único elemento: usa-se vírgula no final

t3 = 10, 20, 30, 40, 50  # se tupla tem mais de 1 elemento,
                         # posso omitir os parênteses

tupla_vazia = tuple()


tupla_mista = ('Pen Drive', 25.90, 'Laptop', 2690)

m1 = ( (1, 2, 3), (4, 5, 6) )    # tupla de tuplas (tupla 2d)

m2 = ( [1, 2, 3], [4, 5, 6] )    # tupla de listas
```

Tupla (3/4)

- **Operações – qualquer coisa que <u>não</u> modifique a tupla**

```
t = ("John", "Yoko", "Julian", "Sean")
t[1]                    # 'Yoko' –  veja que utilizamos [ ] para acessar item.
t[1:3]                  # ('Yoko', 'Julian')
"Paul" in t             # False
type(t)                 # <class 'tuple'>
t.index("Julian")   # 2


# não pode modificar ...
# não existem os métodos append(), insert(), pop(), sort(), inverse() para as tuplas
# nem qualquer outro que modificaria a tupla !!!
t[3]='Paul'
Traceback (most recent call last):
  File "<pyshell>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

Tupla (4/4)

- **E se tiver uma lista dentro da tupla?**
  - Caso um dos elementos da tupla for mutável (ex.: uma lista), esse elemento específico pode ser alterado (mas não removido da tupla).

```
>>> t = (5, [10, 20], 100)
>>> t[1].append(30)          # Ok, posso modificar o conteúdo do 2º elemento...
>>> t
(5, [10, 20, 30], 100)


>>>  t.pop(1)                # Mas não posso remover o 2º elemento na tupla
AttributeError: 'tuple' object has no attribute 'pop'

>>>  t.append([50, 100])     # Não posso inserir um novo elemento na tupla
AttributeError: 'tuple' object has no attribute ‘append’

>>> t[2] = 1000             # Não posso substituir um elemento por outro
TypeError: 'tuple' object does not support item assignment
```

Conceitos Importantes (1/7)

- **Packing e Unpacking de Sequências (1/3)**

  - **PACKING** – se um conjunto de valores separados por vírgula é fornecido, ele será “empacotado” em tupla (mesmo que os valores não estejam envolvidos em parênteses).

```
>>> dados = 2, 4, 6, 8      # “dados” se torna uma tupla
>>> dados
(2, 4, 6, 8)
```

  - **Uso comum**: função que retorna mais de 1 valor (*veremos em uma aula futura*)

```
>>> return a, b      # retorna um único objeto que é a tupla (a, b)
```

pack = de um conjunto de valores para a tupla

Conceitos Importantes (2/7)

- **Packing e Unpacking de Sequências (2/3)**
  - **UNPACKING** – é o “efeito contrário”... o Python pode “desempacotar” uma sequência para uma série de variáveis.

```
>>> pai, mae, filho = ("John", "Yoko", "Sean")    # pai=“John”, mae=“Yoko”, filho=“Sean”

>>> a, b, c, d = range(7, 11)   # a=7, b=8, c=9, d=10

>>> quociente, resto = divmod(10, 3)  # quociente=3, resto=1 (resultado de 10 dividido por 3)
                                      # divimod é uma função que retorna tupla !!!
>>> for x, y in [ (7, 2), (5, 8), (6, 4) ]:
        print(x, y, x + y)
7 2 9
5 8 13
6 4 10
```

unpack = da tupla para variáveis

Conceitos Importantes (3/7)

- **Atribuição Simultânea**
  - A combinação de packing e unpacking forma uma técnica conhecida como atribuição simultânea (também chamada de **tuple assignment**).
  - **Exemplo 1:** x, y, z = 6, 2, 5
    - Como o Python faz?
      - Primeiro, o lado direito da atribuição é automaticamente empacotado em uma tupla.
      - Depois, é automaticamente desempacotado e seus elementos são atribuídos às variáveis “x”, “y” e “z”
      - No final, temos x=6, y=2, z=5
  - **Exemplo 2**: código para trocar o conteúdo de duas variáveis

```
# sem atribuição simultânea
temp = a
a = b
b = temp
```

```
# com atribuição simultânea
a, b = b, a
```

Conceitos Importantes (4/7)

- **Container**

  - **Container**: qualquer objeto que contenha um número arbitrário de elementos.
    - Em nosso curso, utilizamos as palavras **container**, **coleção** e **estrutura de dados** de forma intercambiável.
    - Containers do Python padrão abordados em nosso curso:
      - Lista;
      - Tupla;
      - Dicionário;
      - String
    - Containers que não são vistos nesse curso:
      - Conjunto;
      - coleções do módulo '**collections**'.

Conceitos Importantes (5/7)

- **Sequência**
  - **Sequência**: é o tipo de container onde:
    - (i) Os elementos estão ordenados em uma sequência definida;
    - (ii) Qualquer elemento pode ser acessado através de seu índice.
    - Sequências nativas do Python:
      - Lista;
      - Tupla;
      - String;
      - Objetos range;
      - byte sequences; (*não é coberto nesse curso*)
      - byte arrays. (*não é coberto nesse curso*)

Conceitos Importantes (6/7)

- **Imutável x Mutável**
  - **Container Imutável (*immutable*)**: container em que os elementos possuem um valor fixo na criação que não poderá ser alterado subsequentemente.
    - **Ex.**: tupla.
    - Toda operação de atribuição com um tipo imutável é destrutiva: o conteúdo anterior é destruído e o novo armazenado
    - **IMPORTANTE**: Todos os tipos básicos (int, float, str e bool) também são imutáveis.
  - **Container Mutável (*mutable*)**: container em que os elementos podem ser alterados a qualquer momento.
    - **Ex.**: lista.

Conceitos Importantes (7/7)

- **Valor None**

  - **Valor None**: trata-se de um valor especial, que possui o seu próprio tipo ("NoneType") . Ele pode ser entendido como “ausência de qualquer coisa”.

  - O None é usado que for preciso registrar que um determinado valor é desconhecido ou ausente.

  - Para testar se um valor é None, precisamos usar **is None** (não pode ser == None)

```
# testando se variável contém None
x = None

if x is None:
    print('a variável possui valor None')
else:
    print('a variável armazena alguma coisa')
```

String (1/12)

- **Definição**
  - Uma string em Python é uma **sequência imutável de caracteres**.
    - Por isso, mesmo que str seja um tipo básico, qualquer string pode ser manipulada como uma **tupla** de caracteres (um container!)

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]
E	s	t	a	t	í	s	t	i	c	a

  - Características:
    - **Sequência** – caracteres possuem uma ordem determinada.
    - **Imutável** – nenhum caractere pode ser alterado / inserido / removido.
    - **Sem restrição de comprimento** – o limite é a memória do computador.
    - **Iterável** – capaz de retornar seus caracteres um por vez em um laço.
    - **Objetos str** – são objetos do tipo (classe) str, possuindo vários métodos pré-definidos para sua manipulação.

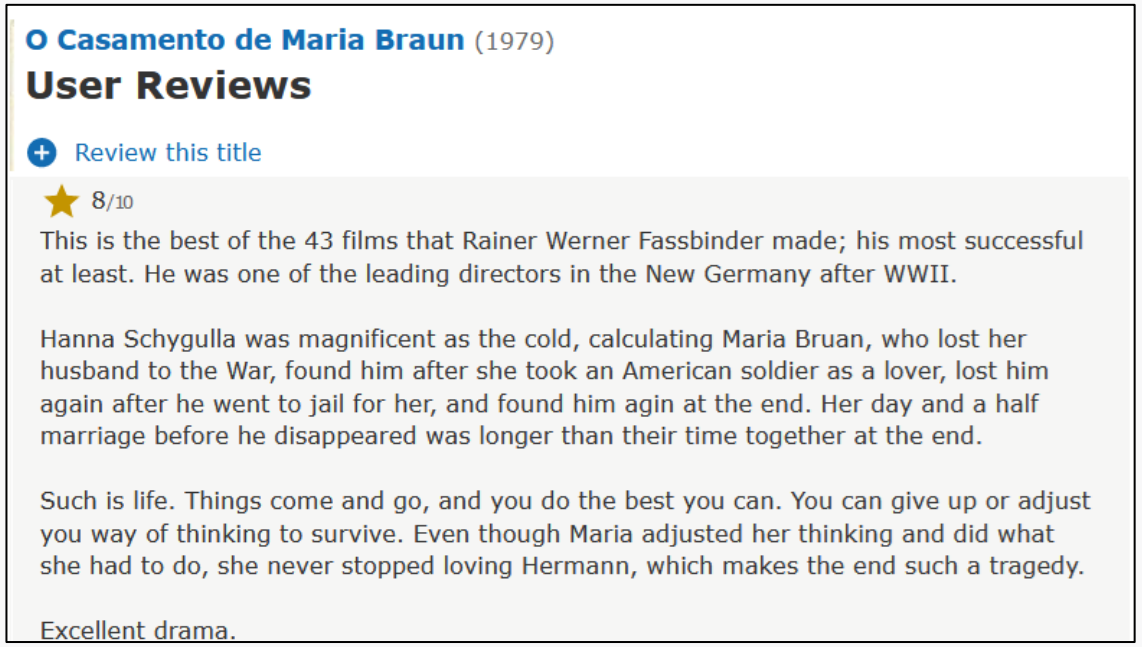
String (2/12)

- **Strings podem representar qualquer coisa.**
  - uma frase*:
    - f = "O sorriso do cachorro tá no rabo..."
  - uma sequência de DNA:
    - s = "CGTAAACTGCTTTAATCAAACGC"
  - um endereço da internet :
    - url = "https://paises.ibge.gov.br/"
  - nomes de pessoas, lugares, coisas, ...
    - poeta = "Mário Quintana“
    - local = "Porto Alegre“
    - objeto = "livro"

* “Toque Frágil” – Walter Franco

String (3/12)

- **Strings podem representar qualquer coisa.**
  - e-mails, postagens de redes sociais, textos de sites da internet, ...



O Casamento de Maria Braun (1979)

User Reviews

Review this title

8/10

This is the best of the 43 films that Rainer Werner Fassbinder made; his most successful at least. He was one of the leading directors in the New Germany after WWII.

Hanna Schygulla was magnificent as the cold, calculating Maria Bruan, who lost her husband to the War, found him after she took an American soldier as a lover, lost him again after he went to jail for her, and found him agin at the end. Her day and a half marriage before he disappeared was longer than their time together at the end.

Such is life. Things come and go, and you do the best you can. You can give up or adjust you way of thinking to survive. Even though Maria adjusted her thinking and did what she had to do, she never stopped loving Hermann, which makes the end such a tragedy.

Excellent drama.

  - bases de dados

```
renda,empregos,sexo,escolaridade
9.90,2,M,superior
6.22,3,F,médio
1.50,1,M,fundamental
0.00,0,F,médio
2.57,1,M,médio
```

String (4/12)

- **Criação de Strings**

  - Pode-se usar aspas simples ou duplas:

    ```
    s1 = 'Estatística'
    s2 = "Processamento de Linguagem Natural"
    ```

  - Uma string definida com aspas duplas pode conter aspas simples (e vice-versa).

    ```
    s3 = 'Vou para a aula de "PROBAB" hoje!'
    ```

  - Strings com **múltiplas linhas** devem ser definidas com o uso de **3 aspas** (simples ou duplas):

    ```
    frase = '''Artificial intelligence (A.I.) is the study of how to make
    computers do things that people are better at.
    Elaine Rich – 1983 '''
    ```

String (5/12)

- **Concatenação – operador +**
  - Produz uma nova string combinando todos os operandos.

```
>>> s1 = "John"
>>> s2 = "Lennon"
>>> s3 = s1 + s2 + " – Imagine"

>>> s3
> John Lennon – Imagine
```

- **Repetição – operador ***
  - Produz uma nova string com o conteúdo de outra repetido *n* vezes.

```
>>> a = "Yoko"
>>> b = a * 3

>>> b
> YokoYokoYoko
```

String (6/12)

- **Relação de Pertinência – operador in**
  - Verifica se uma substring **pertence** a uma string.

```
s = "John Lennon"

"L" in s          # True
"Lennon" in s     # True – pode verificar tanto caractere como substring!
"lennon" in s     # False – diferencia maiúsculas e minúsculas
```

- **Operador not in (faz o oposto)**
  - Verifica se uma substring **não pertence** a uma string

```
s = "John Lennon"

"L" not in s          # False
"Lennon" not in s     # False
"lennon" not in s     # True – "lennon" escrito dessa forma não está em s
```

String (7/12)

- **Funções built-in (pré-definidas na linguagem)**
  - **len(*s*)**: retorna o comprimento da string *s*.
  - **str(*v*)**: retorna a representação string de um valor *v*
  - **isinstance(*v*, str)**: verifica se o valor *v* é valor do tipo str (string).
  - **chr(*n*)**: retorna o caractere *c* associado ao código Unicode* *n.*
  - **ord(*c*)**: retorna o código Unicode *n* associado ao caractere *c*.

```
len("Bossa nova")      # 10

str(49.2)              # “49.2”
str(3 + 29)            # “32”
str("Olá")             # “Olá”

isinstance('samba', str)  # True
isinstance(3.14, str)     # False
```

```
ord("a")       # 97
ord("€")       # 8364
ord("∑")       # 8721


chr(97)        # "a"
chr(8364)      # "€"
chr(8721)      # "∑"
```

* Unicode é um padrão que associa um número inteiro distinto (chamado código Unicode) para cada um dos caracteres e símbolos de todos os idiomas existentes.

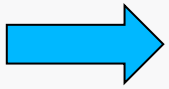
String (8/12)

- **É possível indexar, fatiar e iterar por strings**

- **Iterando com base nos caracteres**

```
nome = "John"

for letra in nome:
    print(letra)
```

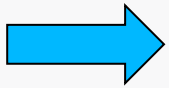


```
J
o
h
n
```

- **Iterando com base nos índices**

```
nome = "John"

for k in range(len(nome)):
    print(f"elemento {k} = {nome[k]}")
```



```
elemento 0 = J
elemento 1 = o
elemento 2 = h
elemento 3 = n
```

String (9/12)

- **Fatiamento (*slicing*):** é igual o das listas e tuplas

[-10]	[-9]	[-8]	[-7]	[-6]	[-5]	[-4]	[-3]	[-2]	[-1]
B	o	s	s	a		n	o	v	a
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]

```
s = "Bossa nova"

s[0:5]      # 'Bossa'
s[:5]       # 'Bossa' (parâmetro i pode ser omitido se for 0)
s[6:10]     # 'nova'
s[6:]       # 'nova' (parâmetro j pode ser omitido se for tamanho de s)
s[2:8]      # 'ssa no'
s[-4:]      # 'nova' (fatia com os quatro últimos caracteres)
s[-9:-5]    # 'ossa'
s[:-6]      # 'Boss'  (dado que len(s) = 10 e j = -6, gera fatia com os
            #          10 – 6 = 4 primeiros caracteres de s)
```

String (10/12)

- **Comparação de strings**
  - Feitas com base no **código Unicode** de cada caractere.
    - A comparação é feita **letra por letra**.
    - Duas strings são **iguais** apenas se armazenam a **mesma palavra, escrita de maneira idêntica** (incluindo maiúsculo/minúsculo e acentos).
    - letras maiúsculas possuem códigos Unicode menores do que as minúsculas ...
    - números têm códigos menores que qualquer letra ...
    - letras acentuadas código maior do que as não acentuadas ...

```
p1 = 'áaa'
p2 = 'aaa'
p3 = 'AAA'
p4 = 'AAA'

p1 == p2    # False
p3 == p4    # True
p2 < p1     # True
p2 < p3     # False
```

- *Consulte a Tab. Unicode*: https://en.wikipedia.org/wiki/List_of_Unicode_characters

String (11/12)

- **Sequências de Escape**
  - A barra invertida “ **** ” dentro de uma string é chamada de **caractere de escape** e possui uma função especial.
    - Ela permite com que seja especificados caracteres especiais como por exemplo nova linha (**\n**) e tabulação (**\t**). Veja alguns exemplos abaixo:

- \' → aspas simples
- \" → aspas duplas
- \\ → barra invertida
- \n → nova linha
- \t → tabulação
- \r → retorno de carro
- \u → código Unicode com valor até 65535
- \U → código Unicode com valor acima de 65535

```
>>> s1 = "samba\nbossa\tnova"
>>>
>>> print(s)
samba
bossa   nova

>>> s2 = 'barra invertida: \\'
>>> print(s2)
barra invertida: \

>>> s3 = '\U0001F642\u2600'
>>> print(s3)
🙂☀
```

String (12/12)

- **Raw Strings**
  - Conforme acabou de ser mostrado, barras invertidas em strings são utilizadas para definir sequências de escape.
    - Por isso, para inserir uma barra precisamos especificar duas:

```
>>> pasta = "C:\\Windows\\temp"
```

  - Em situações como essa podemos usar as **raw strings**.
    - São definidas com o uso da letra **r** precedendo a string.
    - Uma raw string trata a barra invertida como um caractere normal.
    - O Python a converterá automaticamente para uma string normal, com as duas barras. Veja:

```
>>> pasta = r"C:\Windows\temp"
>>> pasta
'C:\\Windows\\temp'
```

Métodos de String (1/16)

- **O que é um Método?**
  - Quando você cria uma string no Python, você "ganha de presente" uma série de métodos, que servem para **executar operações** sobre a string.
    - (*mesma coisa que ocorre com outros containers, como listas e tuplas*)
  - Na realidade, toda string Python é um objeto que contém:
    - **Dados**: os caracteres armazenados; e
    - **Métodos**: as funções para executar tarefas sobre os dados.
  - A sintaxe para executar um método sobre uma string *s* é:
    - **s.metodo(<args>)**
      - Executa o método chamado "metodo()" sobre os dados de *s*.
      - <args> especifica os argumentos passados para o método, caso existam

Métodos de String (2/16)

- A seguir, serão apresentados os principais métodos de strings do Python. Por questões didáticas, eles foram divididos em **5 categorias**:

  I. Conversão Maiúsculo ↔ Minúsculo
  II. Busca de Substrings
  III. Classificação dos Tipos de Caracteres
  IV. Modificação
  V. Conversão String ↔ Lista

- Antes de começar, é **MUITO IMPORTANTE** saber o seguinte:
  - **Nenhum** método **altera a string**, pois a string é **imutável** !!!
  - O máximo que podem fazer é retornar um "**clone modificado**" da string original.

- São **muitos métodos**... mas **na VAE** serão cobrados **apenas esses**:
  - lower(), upper(), count(), find(), index(), split(), replace(), strip(), join(), split().

Métodos de String (3/16)

- **Categ. I – Conversão Maiúsculo ↔ Minúsculo**
  - s.**lower()** : retorna cópia de *s* com todas letras convertidas p / minúsculo.
  - s.**upper()** : retorna cópia de *s* com todas letras convertidas p/ maiúsculo.
  - s.**capitalize()** : retorna cópia de *s* com letra da posição 0 convertida p/ maiúsculo.
  - s.**swapcase()** : retorna cópia de *s* com maiúsculas trocadas p/ minúsculo (e vice-versa)
  - **s.title()**: retorna cópia de *s* com 1ª letra de cada palavra em maiúsculo e demais em minúsculo.

```
s = 'rio de janeiro 40oC'

s.lower()        # 'rio de janeiro 40oc'

s.upper()        # 'RIO DE JANEIRO 40OC'

s.capitalize()   # 'Rio de janeiro 40oc'

s.swapcase()     # 'RIO DE JANEIRO 40Oc'

s.title()        # 'Rio De Janeiro 40Oc'
```

Métodos de String (4/16)

- **<u>Categ. II</u> – Busca de Substrings (1/3)**
  - São os métodos que oferecem formas para **buscar** por uma substring dentro de uma string.
  - Todos suportam dois **parâmetros opcionais**: *i* e *j*.
    - Se indicados, servirão para definir a fatia da string em que os métodos irão atuar.
    - A fatia terá início no caractere da posição *i* e terminará no de posição *j-1*.

Métodos de String (5/16)

- **Categ. II – Busca de Substrings (2/3)**
  - s.**count(*sub, <i>, <j>*)**: conta número de ocorrências da substring *sub* em *s* ou dentro da alguma fatia específica de *s* caso *i* e *j* tenham sido definidos.
  - s.**find(*sub, <i>, <j>*)**: verifica se a substring *sub* ocorre na string *s* ou fatia de *s*. Se *sub* é encontrada, retorna o índice da primeira ocorrência. Senão retorna -1;
  - s.**rfind(*sub, <i>, <j>*)**: igual ao find(), mas verifica de trás pra frente.
  - s.**index(*sub, <i>, <j>*)**: igual ao find(), mas dispara exceção caso *sub* não seja encontrada.
  - s.**rindex(*sub, <i>, <j>*)**: igual ao index(), mas verifica de trás para frente.
  - s.**endswith(*suf, <i>, <j>*)**: verifica se *s* ou uma fatia de *s* termina com o sufixo *suf*.
  - s.**startswith(*pre, <i>, <j>*)**: verifica se *s* ou uma fatia de *s* começa com o prefixo *pre*.

Métodos de String (6/16)

- **Categ. II – Busca de Substrings (3/3)**

```
frase = "O sorriso do cachorro tá no rabo..."

frase.count("rr")                 # 2
frase.count("rr", 18, 20)         # 1
frase.count("gato")               # 0

frase.find("rr")                  # 4
frase.rfind("rr")                 # 18
frase.find("xa")                  # -1 (substring não encontrada)
frase.index("xa")                 # dispara exceção (substring não encontrada)

frase.startswith("O sorriso")     # True
frase.endswith("rabo...")         # True
frase.startswith("o")             # False, pois a string começa com “O” e não “o”
frase.endswith("iso", 6, 9)       # True
```

Métodos de String (7/16)

- **<u>Categ. III</u> – Classificação dos Tipos de Caracteres (1/2)**
  - s.**isalnum()** : retorna True se *s* não é vazio e possui apenas números e letras.
  - s.**isalpha()** : retorna True se *s* não é vazio e possui apenas letras.
  - s.**isdigit()** : retorna True se *s* não é vazio e possui apenas números.
  - s.**islower()** : retorna True se *s* não é vazio e todas as letras que possuir estiverem em minúsculo.
  - s.**isupper()** : retorna True se *s* não é vazio e todas as letras que possuir estiverem em maiúsculo.
  - s.**isspace()** : retorna True se *s* não é vazio e todos os seus caracteres são do tipo whitespace, tais como: "\n", "\t", " " (espaço em branco), entre outros.
  - s.**istitle()** : retorna True se *s* não é vazio e a primeira letra de cada palavra está em maiúsculo.
  - s.**isprintable()** : retorna True se *s* está vazio ou todos os caracteres contidos em *s* são “visíveis”. Exemplo: "\n“ e "\t“ não são visíveis, mas " " é.
  - s.**isidentifier()** : retorna True se *s* é um identificador válido do Python, de acordo com as regras da linguagem (ou seja, um nome que poderia ser dado para uma variável, classe ou função).

Métodos de String (8/16)

- **Categ. III – Classificação dos Tipos de Caracteres (2/2)**

```
"UB40".isalnum()            # True
"UB-40".isalnum()           # False (possui traço "-", que não é letra nem número)
"".isalnum()                # False (string vazia)
"Jonh".isalpha()            # True
"Jonh123".isalpha()         # False (possui números)
"Jonh Lennon".isalpha()     # False (possui espaço em branco, que não é letra)
"54321".isdigit()           # True
"54321 fogo!!!".isdigit()   # False (possui espaço, exclamação e letras)
"54321 fogo!!!".islower()   # True (as letras existentes estão todas em minúsculo)
"UB40".isupper()            # True (as letras existentes estão todas em maiúsculo)
"AbC1$D".isupper()          # False (letra "b"" não é maiúscula)
"Girl From Ipanema".istitle()   # True (letras iniciais de cada palavra em maiúsculo)
"Garota de Ipanema".istitle()   # False ("de" não começa com letra maiúscula)
```

Métodos de String (9/16)

- **Categ. IV – Formatação e Modificação (1/6)**
  - No Python, as **strings** são objetos **imutáveis**.
  - Isso significa que, depois de criadas:
    - Não podem ter caracteres modificados.
    - Não podem ter caracteres removidos.
    - Não podem ter caracteres inseridos.
  - Qualquer tentativa de modificar a string resultará em erro:

```
>>> f = "vida imutável"
>>> f[0] = "V"

Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
```

Métodos de String (10/16)

- **Categ. IV – Formatação e Modificação (2/6)**
  - As única coisa que pode ser feita é:
    - **Recriar** a string a partir de uma **cópia modificada**.
  - Isso pode ser feito, por exemplo, usando fatiamento ou algum método de string.

```
>>> f = "vida imutável“

# no exemplo abaixo, “clone1” é uma cópia modificada de f
>>> clone1 = "V" + f[1:]
>>> clone1
'Vida imutável’

# no exemplo abaixo, “f” será destruída e recriada com novo conteúdo
# (ou seja, “f” não é alterada, mas sim destruída e recriada)
>>> f = f.replace("imutável", "bela")
>>> f
'vida bela'
```

Métodos de String (11/16)

- **<u>Categ. IV</u> – Formatação e Modificação (3/6)**
  - s.**center(*n*, <*c*>)** : retorna cópia centralizada de *s* com comprimento *n*. A área em volta terá espaço “ ” ou o caractere *c* caso este seja especificado.
  - s.**ljust(*n*, <*c*>)** : idem, porém a cópia será alinhada à esquerda.
  - s.**rjust(*n*, <*c*>)** : idem, porém a cópia será alinhada à direita.
  - s.**expandtabs(<*tabsize=n*>)** : retorna cópia de *s* com tabulação “\t” substituída por *n* espaços em branco “ ” (8 espaços, caso *tabsize* não seja especificado)
  - s.**strip(<*sub*>)** : retorna cópia de *s* com whitespaces à esquerda e direita removidos. Ou com a substring *sub* removida, se essa for especificada.
  - s.**lstrip(<*sub*>)** : idem, porém remove apenas à esquerda de *s.*
  - s.**rstrip(<*sub*>)** : idem, porém remove apenas à direita de *s.*
  - s.**zfill(*n*)** : retorna cópia de *s* preenchida com zeros à esquerda até o comprimento *n.*
  - s.**replace(*sub_ant, sub_nova,* <*n*>)** : retorna cópia de *s* com todas as ocorrências da substring *sub_ant*, substituídas pela substring *sub_nova*. Se o parâmetro *n* for indicado, fará no máximo *n* trocas.
  - s.**translate(*mapa_traducao*)** : permite a “traduzir” um conjunto de caracteres por outro (*mais detalhes no exemplo!*)

Métodos de String (12/16)

- **Categ. IV – Formatação e Modificação (4/6)**

```
"samba".center(10)          # '  samba   '
"samba".center(10, '+')     # '++samba+++'
"Bossa nova".rjust(15)      # '     Bossa nova'
"rock".ljust(7, "!")        # 'rock!!!'

"      https://www.ibm.com    ".strip()      # 'https://www.ibm.com'
"https://www.ibm.com".lstrip("/:pths")      # 'www.ibm.com'

"12\t34\t56".expandtabs()    # '12      34      56'
"12\t34\t56".expandtabs(4)   # '12  34  56'
"-99".zfill(8)               # '-0000099'

"nova bossa".replace("nova", "new")     # 'new bossa'
"a1a2a3a4a5a6".replace("a", "b", 3)     # 'b1b2b3a4a5a6'
```

Métodos de String (13/16)

- **Categ. IV – Formatação e Modificação (5/6)**
  - Métodos **translate()** e **maketrans()**: podem ser usados em conjunto para traduzir um conjunto de caracteres de uma string para outro conjunto.
  - **Sintaxe: translate(**mapa_de_tradução**)**
    - O **mapa de tradução** é criado com o método **maketrans() :**
      - **maketrans(**from_chars, to_chars, delete_chars**)**
        - **from_chars:** caracteres que serão substituídos (ex.: ‘áéíóú’)
        - **to_chars:** caracteres que entrarão no lugar (ex.: ‘aeiou’)
        - **delete_chars:** caracteres que serão apagados
  - **Exemplo – eliminando os símbolos e os acentos de uma frase:**

```
>>> s = "Olá, Alô!!??"
>>> s = s.translate(str.maketrans("áô", "ao", "!?,"))
>>> s
'Ola Alo'
```

Métodos de String (14/16)

- **Categ. IV – Formatação e Modificação (6/6)**
- **Exemplo: Convertendo para minúsculo, removendo trailing spaces e símbolos e trocando letras com acento por letras sem acentuação.**

```
#suponha que o texto abaixo seja um tweet
>>> texto = " Diagnóstico... não há como você PERDER!!!  "
>>> texto = texto.lower().strip() # conv. p/ minúsc. e remove brancos à direita e esq.

#OP. 1: se quiser eliminar símbolos, faça isso
>>> simbolos = '!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~’
>>> texto.translate(str.maketrans("", "",  simbolos))
'diagnóstico não há como você perder'

#OP. 2: se quiser eliminar os símbolos e trocar letras acentuadas por sem acento
>>> simbolos = '!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~'
>>> traducao = ('àáâãéêíóôõúç','aaaaeeiooouc')
>>> texto.translate(str.maketrans(traducao[0], traducao[1],  simbolos))
'diagnostico nao ha como voce perder'
```

Métodos de String (15/16)

- **<u>Categ. V</u> – Conversão strings ↔ listas (1/2)**
  - s.**join(*w*)** : retorna uma string contendo todos os elementos da lista (ou outra coleção) *w* separados por *s*.
  - s.**partition(*sep*)** : dado um caractere *sep*, gera uma tupla contendo 3 elementos: (substring à esquerda da 1ª ocorrência de *sep*, *sep*, substring à direita de *sep*).
  - s.**rpartition(*sep*)** : idem, porém considera a última ocorrência de *sep*.
  - s.**split(*sep, maxsplit=n*)** : gera uma lista contendo as substrings de *s* delimitadas pelo caractere separador *sep.* Caso *sep* não seja especificado, usa branco “ ”. Se um valor *n* for especificado para o parâmetro maxsplit, considera apenas as *n* primeiras ocorrências de *sep.*
  - s.**rsplit(*sep, maxsplit=n*)** : idem, porém se *maxsplit* for especificado, considera apenas as *n* últimas ocorrências de *sep*.
  - s.**splitlines(*keepends*=False)** : divide *s* em uma lista de linhas. Se o parâmetro *keepends* for especificado como True, mantém o caractere de quebra de linha no final de cada item na lista.

Métodos de String (16/16)

- **Categ. V – Conversão strings ↔ listas (2/2)**

```
"-".join(["John", "Paul", "George", "Ringo"])    # 'John-Paul-George-Ringo'

":".join("Yoko")    # 'Y:o:k:o'

"Jonh@@Paul@@George@@Ringo".partition("@@")
                                    # ('Jonh', '@@', 'Paul@@George@@Ringo')
"Jonh@@Paul@@George@@Ringo".rpartition("@@")
                                    # ('Jonh@@Paul@@George', '@@', 'Ringo')
"Jonh@@Paul@@George@@Ringo".split("@@")  # ['Jonh', Paul', 'George', 'Ringo']

"A hard day's night".split()            # ['A', 'hard', "day's", 'night']

"A\nhard\nday's\nnight".splitlines()    # ['A', 'hard', "day's", 'night']
"A\nhard\nday's\nnight".splitlines(True)    # ['A\n', 'hard\n', "day's\n", 'night']
```