Python – Cadeias de Caracteres: Strings

Prof. André Backes

Definição

- **Cadeia de caracteres ou String**
  - Sequência de caracteres adjacentes na memória
  - Permite representar palavras ou frases dentro do computador
  - Em outras palavras, strings são listas. A diferença é que cada posição da lista contém um único caractere

Definição

- Na inicialização de uma string podemos usar "**aspas duplas**" ou '**aspas simples**'
- O tipo de uma string é a classe **str**

```
>>> texto = "Python"
>>> print(texto)
Python
>>> type(texto)
<class 'str'>
>>>
>>> texto = 'Python'
>>> print(texto)
Python
>>> type(texto)
<class 'str'>
```

Definição

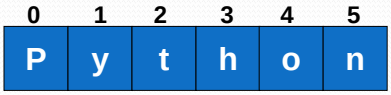
- Podemos utilizar 3 aspas simples na inicialização de uma string. Neste caso, será possível criar uma string contendo mais de uma linha
  - As quebras de linha também serão armazenadas dentro da string.

```
>>> texto = '''Aprender Python
é muito
fácil!'''
>>> print(texto)
Aprender Python
é muito
fácil!
>>> type(texto)
<class 'str'>
```

Acessando seus elementos

- Podemos tratar uma string como uma entidade única
- Mas também podemos acessar seus caracteres individualmente usando colchetes e o índice da posição

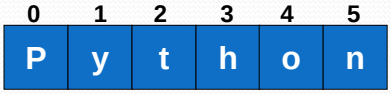
```
>>>
>>> texto = 'Python'
>>> texto[0]
'P'
>>> texto[1]
'y'
>>> texto[5]
'n'
>>>
```



Acessando seus elementos

- Tamanho da string
  - A função **len()** retorna o tamanho de uma string
  - Neste caso, a função retornará 6, que é o número de caracteres na palavra

```
>>>
>>> texto = 'Python'
>>> len(texto)
6
>>>
```

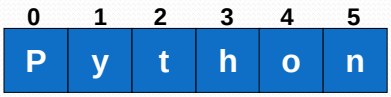



```
>>> texto[6]
Traceback (most recent call last):
  File "<pyshell#46>", line 1, in <module>
    texto[6]
IndexError: string index out of range
>>>
```

Acessando seus elementos

- Não podemos acessar um índice da string que seja maior ou igual ao tamanho da string
  - Os índices dos caracteres de uma string sempre começam em ZERO e vão até TAMANHO-1

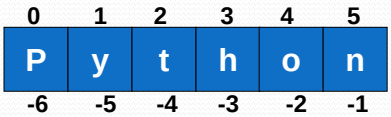
```
>>>
>>> texto = 'Python'
>>> len(texto)
6
>>>
>>> texto[6]
Traceback (most recent call last):
  File "<pyshell#46>", line 1, in <module>
    texto[6]
IndexError: string index out of range
>>>
```



Acessando seus elementos

- Podemos utilizar índices negativos para acessar os caracteres de uma string
- Neste caso, a contagem começa do último caractere da string

```
>>> texto = 'Python'
>>> texto[5]
'n'
>>> texto[-1]
'n'
>>> texto[-2]
'o'
>>> texto[-6]
'P'
```



Acessando seus elementos

- Como nas listas, as strings também suportam acesso a sub-strings ou sub-cadeias de caracteres
  - texto[i:j]
    - seleciona a sub-cadeia dos índices i até j-1
  - texto[i:]
    - seleciona a sub-cadeia dos índice i até o final
  - texto[:j]
    - seleciona a sub-cadeia do início até o índice j-1
  - texto[i:j:k]
    - seleciona a sub-cadeia dos índices i até j-1, indo de k em k
    - i, i+k, i+2k, ..., j-1

Acessando seus elementos

- Selecionando sub-strings ou sub-cadeias de caracteres

```
>>> texto = "Aprender Python é muito fácil!"
>>> texto[9:15]
'Python'
>>> texto[9:]
'Python é muito fácil!'
>>> texto[:15]
'Aprender Python'
>>> texto[0:15:2]
'Arne yhn'
>>>
```

Percorrer uma string

- Podemos percorrer uma string de duas formas
  - Usando um ciclo sobre os índices

```
texto = "Python"

for i in range(len(texto)):
    print(texto[i])
```

Saída

```
>>>
P
y
t
h
o
n
>>>
```

  - Usando um ciclo sobre a sequência

```
texto = "Python"

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

Concatenação de string

- Podemos unir/concatenar duas string para formar uma nova de duas maneiras
  - Usando o operador de soma "+"
  - Separando as strings por **vírgula** no momento da impressão (um espaço será automaticamente inserido entre elas)

```
>>>
>>> texto = "Python"
>>> texto = "Linguagem" + texto
>>> texto
'LinguagemPython'
>>>
```

```
>>>
>>> texto = "Python"
>>> print("Linguagem", texto)
Linguagem Python
>>>
```

Concatenação de string

- Podemos acessar os caracteres individualmente de uma string, mas não podemos modificá-los
  - Felizmente, podemos construir uma outra string via concatenação

```
>>> texto = 'Teste'
>>> texto[0] = 'L'
Traceback (most recent call last):
  File "<pyshell#28>", line 1, in <module>
    texto[0] = 'L'
TypeError: 'str' object does not support item assignment
>>>
>>> texto = 'L' + texto[1:]
>>> print(texto)
Leste
```

Sequências de escape

- Também chamados de códigos de barra invertida
- Permitem o envio de caracteres de controle não gráficos para dispositivos de saída

Código	Comando
\n	nova linha
\t	tabulação
\v	tabulação vertical
\b	retrocesso
\'	aspas simples
\"	aspas duplas
\\	barra invertida

```
>>>
>>> print("Linguagem \n Python")
Linguagem
 Python
>>>
>>> print("Linguagem \t Python")
Linguagem        Python
>>>
>>> print("Linguagem \\ Python")
Linguagem \ Python
>>>
```

Sequências de escape

- As sequências de escape são executas sempre que uma barra invertida é encontrada
- Para evitar que as sequências de escape funcionem, basta definir a string como uma ***Raw String***
  - Para isto basta preceder a string com **r** ou **R**

```
>>> print("Linguagem \\ Python")
Linguagem \ Python
>>>
>>> print(r"Linguagem \\ Python")
Linguagem \\ Python
>>>
```

Formatação de strings

- Também é possível realizar a formatação de string utilizando o operado %
- Forma geral
  - string-a-ser-formatada % (lista-de-valores)
- Todo conteúdo da string da esquerda precedido por um % é substituído por um valor a direita (entre parênteses)

```
>>>
>>> str = "O reajuste foi de %d %% e a inflação de %.2f %%" % (10, 6.5)
>>> str
'O reajuste foi de 10 % e a inflação de 6.50 %'
>>>
```

Formatação de strings

- Na string da esquerda, o conjunto de caracteres depois do % define o tipo de formatação a ser executada

Código	Tipo de formatação
%c	caractere
%s	string
%d	inteiro
%u	inteiro sem signal
%f	reais (ponto flutuante)
%.Nf	reais com N casas decimais
%%	símbolo de %

```
>>>
>>> str = "O reajuste foi de %d %% e a inflação de %.2f %%" % (10, 6.5)
>>> str
'O reajuste foi de 10 % e a inflação de 6.50 %'
>>>
```

Manipulando strings

- Podemos procurar uma **string menor** dentro de uma **string maior**
  - Ex: palavra dentro de uma frase
- Para isso, basta utilizar o operador **in**. Forma geral
  - string1 in string2
- O resultado será **True** se a string1 existir dentro da string2, e **False**, caso contrário

Manipulando strings

- Procurando uma **string menor** dentro de uma **string maior**
  - No caso, as string devem estar escritas exatamente iguais

Exemplos

```
texto = "Aprender Python é muito fácil!"
if "Python" in texto:
    print("Sequencia encontrada!")
else:
    print("Sequencia nao encontrada!")


if "PYTHON" in texto:
    print("Sequencia encontrada!")
else:
    print("Sequencia nao encontrada!")
```

Saída

```
>>>
Sequencia encontrada!
Sequencia nao encontrada!
>>>
```

Manipulando strings

- Também podemos utilizar os operadores relacionais (==, !=, <, <=, >, >=) para comparar duas string
  - A comparação é feita usando a ordem lexicográfica (i.e. ordem do dicionário)

Exemplo

```
nome1 = "Ricardo"
nome2 = "Ana"

if nome1 < nome2:
    print(nome1,"vem antes de ",nome2)
else:
    print(nome2,"vem antes de ",nome1)
```

Saída

```
>>>
Ana vem antes de  Ricardo
>>>
```

Manipulando strings

- Nas comparações, letras maiúsculas e minúsculas são consideradas diferentes

Exemplos

```
nome1 = "Ricardo"

if nome1 == "Ricardo":
    print("Nomes iguais!")
else:
    print("Nomes diferentes!")
```

Saída

```
>>>
Nomes iguais!
>>>
```

```
nome1 = "Ricardo"

if nome1 == "RICARDO":
    print("Nomes iguais!")
else:
    print("Nomes diferentes!")
```

```
>>>
Nomes diferentes!
>>>
```

Manipulando strings

- Essa diferença entre maiúsculas e minúsculas ocorre pois a comparação é feita pelos códigos numéricos dos caracteres
  - Função **ord()**: código numérico de um caractere
  - Função **chr()**: caractere de um código numérico

```
>>> ord("A")
65
>>> ord("B")
66
>>> ord("a")
97
```

```
>>> chr(75)
'K'
>>> chr(83)
'S'
>>> chr(97)
'a'
```

Métodos sobre strings

- Uma string é uma classe e, portanto, possui diversos métodos já definidos
  - Um dos jeitos mais simples de manipular strings é utilizar os métodos que já fazem parte da string
- Esses métodos permitem executar diversas tarefas
  - Conversão maiúsculo/minúsculo, localizar e substituir substrings, etc
  - Esses métodos nunca modificam o conteúdo original

```
>>> texto = "Aprender Python é muito fácil!"
>>> texto.upper()
'APRENDER PYTHON É MUITO FÁCIL!'
>>> texto
'Aprender Python é muito fácil!'
```

Métodos sobre strings

- Forma geral de uso dos métodos
  - **Objeto-string.nome-método()**
- Alguns métodos
  - lower(): converte para minúsculo
  - upper(): converte para maiúsculo
  - replace(c1,c2): troca os caracteres **c1** por **c2**
  - strip(): remove espaços do início e fim
  - split(): separa uma string por espaços e devolve uma lista de strings
  - split(ch): separa uma string usando o caractere **ch** e devolve uma lista de strings

Métodos sobre strings

- Exemplos

```
>>>
>>> texto = "Aprender Python é muito fácil!"
>>> texto.lower()
'aprender python é muito fácil!'
>>> texto.upper()
'APRENDER PYTHON É MUITO FÁCIL!'
>>> texto.split()
['Aprender', 'Python', 'é', 'muito', 'fácil!']
>>> texto.split('é')
['Aprender Python ', ' muito fácil!']
>>> texto.replace(' ','-')
'Aprender-Python-é-muito-fácil!'
>>>
```