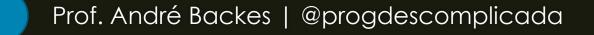
Strings



Definição

Cadeia de caracteres ou String

- O 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'>
```

- 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'
>>>
```



- 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
>>>
```

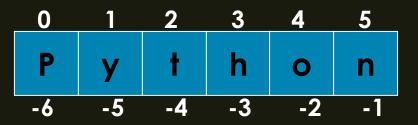


- Não podemos acessar um índice da string que seja maior ou igual ao tamanho da string
 - Os índices dos caracteres de uma string sempre começam em ZERO e vão até TAMANHO-1

```
>>> texto = 'Python'
>>> len(texto)
>>>
>>> texto[6]
Traceback (most recent call last):
  File "<pyshell#46>", line 1, in <module>
    texto[6]
IndexError: string index out of range
       0
```

- 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'
```



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

 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'
>>>
```

Percorrendo uma string

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

```
texto = 'Python'
# Percorre apenas os elementos
for letra in texto:
    print(letra)
# Percorre os índices e elementos
for indice in range(len(texto)):
    print(texto[indice])
>>>
```

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'
>>>
>>>
>>>
>>>
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
```

Repetição de string

 Podemos criar uma nova string a partir da repetição de outra string usando o operador de *

```
>>> str = "a"
>>> str1 = str * 3
>>> str1
    'aaa'
>>>
>>> str = 'abc'
>>> str1 = str * 3
>>> str1
    'abcabcabc'
>>>
```

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 |
| \ † | tabulação |
| \v | tabulação vertical |
| \b | retrocesso |
| \' | aspas simples |
| \" | aspas duplas |
| \\ | barra invertida |

Sequências de escape

- As sequências de escape são executas sempre que uma barra invertida é encontrada
- Alguns exemplos de uso

Sequências de escape

- 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 sinal |
| %f | reais (ponto flutuante) |
| %.Nf | reais com N casas decimais |
| %% | símbolo de % |

Formatação de strings

- Outra forma de combinar texto e valores formatados é usando strings literais formatadas.
 - Para isto basta preceder a string com f ou F
- Assim, podemos especificar o nome da variável a ser escrita entre chaves

```
idade = 25
altura = 1.76
texto = f'A altura é {altura} e a idade é {idade}'
print(texto)
```

- O Podemos procurar uma string menor dentro de uma string maior
 - Exemplo: 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
 - Neste caso, as string devem estar escritas exatamente iguais

Exemplo

```
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!
>>>
```

- Também podemos utilizar os operadores relacionais (==, !=, <, <=, >, >=) para comparar duas string
 - A comparação é feita usando a ordem lexicográfica (i.e. ordem do dicionário)
 - Nas comparações, letras maiúsculas e minúsculas são consideradas diferentes

Exemplo

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

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

Saída

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

Exemplo

```
nome1 = "Ricardo"

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

else:
    print("Nomes diferentes!")

nome1 = "Ricardo"

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

else:
    print("Nomes diferentes!")
```

Saída

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

>>>

Nomes diferentes!
>>>
```

- 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

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
 - o strip(): remove espaços do início e fim
 - o split(): separa uma string por espaços e devolve uma lista de strings
 - o 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!'
>>>
```

Material Complementar

- Vídeo Aulas
 - O Aula 24 String
 - O https://youtu.be/D7GD5EY0ctg
 - O Aula 25 String: concatenação e criação de sub-string
 - https://youtu.be/eKytKeN6kGU
 - Aula 26 String: formatação
 - O https://youtu.be/wM4HfWhNu_0
 - O Aula 27 String: operações e métodos
 - O https://youtu.be/tSALV1GhQyU