Introdução à Programação para Sensoriamento Remoto

Aula 08 – Introdução à Programação com a Linguagem Python

Gilberto Ribeiro de Queiroz
Thales Sehn Körting
Fabiano Morelli

03 de Abril de 2019

Tópicos

- Strings.

Strings

Tipo `str`: Sequência de Caracteres

- O tipo `str` é capaz de representar uma **sequência de caracteres** ou **cadeia de caracteres** ou **string**.

- Uma **string** pode ser expressa entre **aspas duplas** ou entre **aspas simples**:

```
print("Gilberto")
```

```
print('Gilberto')
```

```
print("""Gilberto
         Ribeiro de
         Queiroz""")
```

Concatenação de Strings: `s + t`

- O **operador+** para strings possui uma interpretação diferente dos tipos numéricos, ele realiza a **concatenação de duas strings**:

```
>>> "Gilberto" + "Ribeiro"
'GilbertoRibeiro'

>>> "Gilberto " + "Ribeiro"
'Gilberto Ribeiro'

>>> "Gilberto " + "Ribeiro " + "de " + "Queiroz"
'Gilberto Ribeiro de Queiroz'
```

Concatenação de Strings: `s + t`

- Uma string é uma **sequência imutável** de caracteres.

- A concatenação de strings gera um novo objeto, isto é, uma nova string.

- Portanto, cuidado ao utilizar o **operador`+`** repetidamente pois ele pode gerar um *overhead* desnecessário no seu programa*.

*Há outras opções de realizar operações semelhantes, como por exemplo, usando o operador `str.join()` ou `io.StringIO`.

Repetição de Strings: n * s

- Quando aplicado a um número inteiro **n** e uma string **s**, o **operador*** funciona como se estivéssemos adicionando a string **s** a ela própria, **n** vezes*:

```
>>> "Gilberto" * 3
'GilbertoGilbertoGilberto'

>>> 4 * "Gilberto"
'GilbertoGilbertoGilbertoGilberto'

>>> -4 * "Gilberto"
''
```

Para valores negativos, a expressão é equivalente a `0 * s`.

*A lógica é que assim como => `3 * 2 = 2 + 2 + 2` => `3 * "a" = "a" + "a" + "a"`

Pertinência: `x in s`

- **`x in s`:** retorna verdadeiro se a string **`s`** contém a sequência **`x`**.

```
>>> "i" in "Gilberto Ribeiro de Queiroz"
True

>>> "a" in "Gilberto Ribeiro de Queiroz"
False

>>> "Ri" in "Gilberto Ribeiro de Queiroz"
True

>>> "Rie" in "Gilberto Ribeiro de Queiroz"
False
```

Impertinência: `x not in s`

- **`x not in s`:** retorna falso se a string **`s`** contém a sequência **`x`**.

```
>>> "i" not in "Gilberto Ribeiro de Queiroz"
False

>>> "a" not in "Gilberto Ribeiro de Queiroz"
True

>>> "Ri" not in "Gilberto Ribeiro de Queiroz"
False

>>> "Rie" not in "Gilberto Ribeiro de Queiroz"
True
```

Comprimento da cadeia: `len(s)`

- O operador **`len(s)`** retorna o número de caracteres (ou tamanho) da string **`s`**.

```
>>> len("Gilberto Ribeiro de Queiroz")
27

>>> len("")
0
```

Índice (*indexing*): **s[i]**

- O **operador[]** retorna o *i-th* caractere da string **s**.

```
>>> "Gilberto Ribeiro de Queiroz"[3]
'b'

>>> "Gilberto Ribeiro de Queiroz"[0]
'G'

>>> "Gilberto Ribeiro de Queiroz"[-1]
'z'

>>> "Gilberto Ribeiro de Queiroz"[28]
IndexError: string index out of range
```

Obs.: 1. O primeiro elemento possui índice 0.
2. Um índice fora do limite irá lançar uma exceção do tipo `IndexError`.

Slicing: `s[i:j]`

- O **operador**`[i:j]` extrai substrings de tamanho arbitrário.

```
>>> "Gilberto Ribeiro de Queiroz"[0:3]
'Gil'

>>> "Gilberto Ribeiro de Queiroz"[:-24]
'Gil'

>>> "Gilberto Ribeiro de Queiroz"[9:]
'Ribeiro de Queiroz'

>>> "Gilberto Ribeiro de Queiroz"[:]
'Gilberto Ribeiro de Queiroz'
```

Se o valor antes do ":" for omitido, subentende-se o valor `0`.

Se o valor após ":" for omitido, subentende-se o valor `len(s)`.

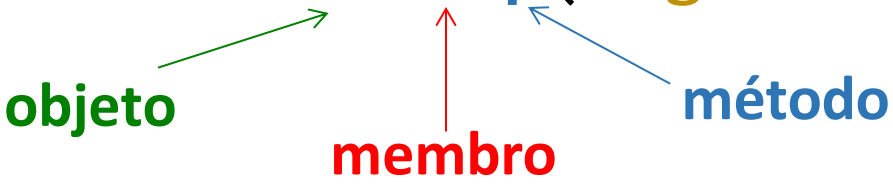
Obs.: A lógica do interval aberto à direita é que seja possível fazer: `s[0:len(s)]`

Tipo `str`: Operações sobre Strings

- Diversas operações sobre o tipo string encontram-se na forma de **métodos**.

- Um **método** nada mais é do que uma notação especial para chamada de funções, onde a função irá operar sobre o próprio dado indicado na chamada.

- A sintaxe dessas operações, em geral, serão da forma: `str.op(argumentos)`



s.find(sub[, start[, end]])

- Procura a ocorrência da substring `sub` em `s`.
- Retorna `-1` caso não encontre uma ocorrência.
- Retorna a posição do primeiro caractere na string onde a ocorrência da substring é encontrada.

```
>>> "Gilberto Ribeiro".find("rto")
5
>>> "Alberto Queiroz".find("rto")
4
>>> "Cassia Diniz".find("rto")
-1
```

Obs.: Se você não precisar saber a posição da substring, use o operador `in`.

s.join(iterable)

- Retorna uma string formada pela concatenação de uma sequência de strings (iterable).

```
>>> "-".join( ("Gilberto", "Ribeiro", "de", "Queiroz") )
'Gilberto-Ribeiro-de-Queiroz'

>>> " ".join( ("Gilberto", "Ribeiro", "de", "Queiroz") )
'Gilberto Ribeiro de Queiroz'

>>> "".join( ("Gilberto", "Ribeiro", "de", "Queiroz") )
'GilbertoRibeirodeQueiroz'
```

Obs.: A sequência de strings separadas por vírgula delimitadas por "(" e ")" é uma tupla.

s.split(sep=None, maxsplit=1)

- Retorna uma **lista** de palavras usando o separador **sep** como delimitador.

```
>>> "Gilberto Ribeiro de Queiroz".split()
['Gilberto', 'Ribeiro', 'de', 'Queiroz']

>>> "Gilberto-Ribeiro-de-Queiroz".split("-")
['Gilberto', 'Ribeiro', 'de', 'Queiroz']

>>> "1,2,,3,".split(",")
['1', '2', '', '3', '']
```

Obs.: A sequência de strings separadas por vírgula delimitadas por "[" e "]" é uma lista.

s.replace(old, new[, count])

- Retorna uma string onde toda a ocorrência da substring **old** em **s**, é substituída pela substring **new**.
- O argumento opcional **count** limita o número de substituições.

```
>>> "Gilberto Ribeiro de Queiroz".replace("i", "@")
'G@lberto R@be@ro de Que@roz'

>>> "Gilberto Ribeiro de Queiroz".replace("i", "@", 2)
'G@lberto R@beiro de Queiroz'
```

Tipo `str`: Outras Operações

- `str.isdigit()`:
  Retorna `True` se todos os caracteres da string são dígitos, caso contrário, retorna `False`.

- `str.islower()`:
  Retorna `True` se todos os caracteres são minúsculos e exista pelo menos uma letra, caso contrário, retorna `False`.

- `str.isupper()`:
  Retorna `True` se todos os caracteres são maiúsculos e exista pelo menos uma letra, caso contrário, retorna `False`.

- `str.lower()`:
  - Retorna uma **cópia** da string com todos os caracteres em minúsculo.

- `str.upper()`:
  - Retorna uma **cópia** da string com todos os caracteres em maiúsculo.

Obs.: Existem diversas operações sobre strings, consulte a documentação oficial do Python.

Tipo `str`: Considerações

- Nas versões Python 2, a comparação entre um número e uma string através dos operadores relacionais (ou de comparação) é uma expressão válida. No entanto, a checagem de tipos de Python 3 é um pouco mais estrita, avaliando esse tipo de construção como um erro.

- As strings são apenas um dos tipos de sequência disponíveis em Python*. Por conta disso, as strings compartilham diversas operações com os outros tipos.

- Strings são **sequências imutáveis**.

***** Tuplas, Listas, e Ranges são outros tipos de sequência.

Considerações Finais

Considerações Finais

- O tipo string em Python é muito poderoso, fornecendo diversas operações.

- O tipo string é um dos tipos de sequência existentes na linguagem Python:
  - Uma string é uma sequência imutável.
  - Existem diversas operações comuns entre os tipos de sequência.

- Em aulas específicas iremos estudar as expressões regulares e a formatação de strings.

Referências Bibliográficas

Referências Bibliográficas

- Common Sequence Operations. Acesso: Março, 2018.

- String Methods. Acesso: Março, 2018

- Text Sequence Type – str. Acesso: Março, 2018.