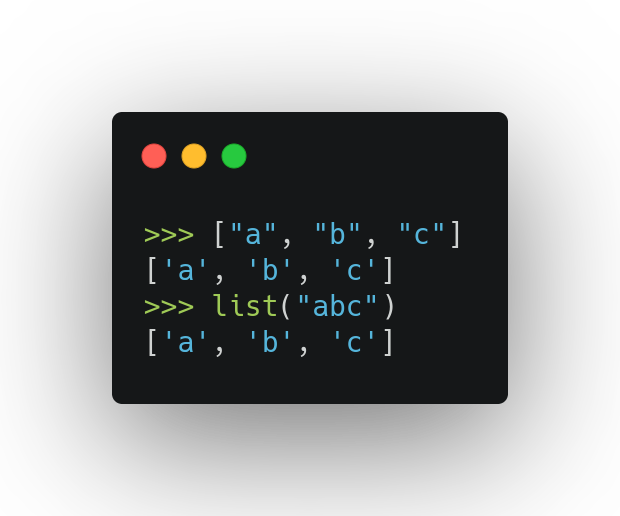
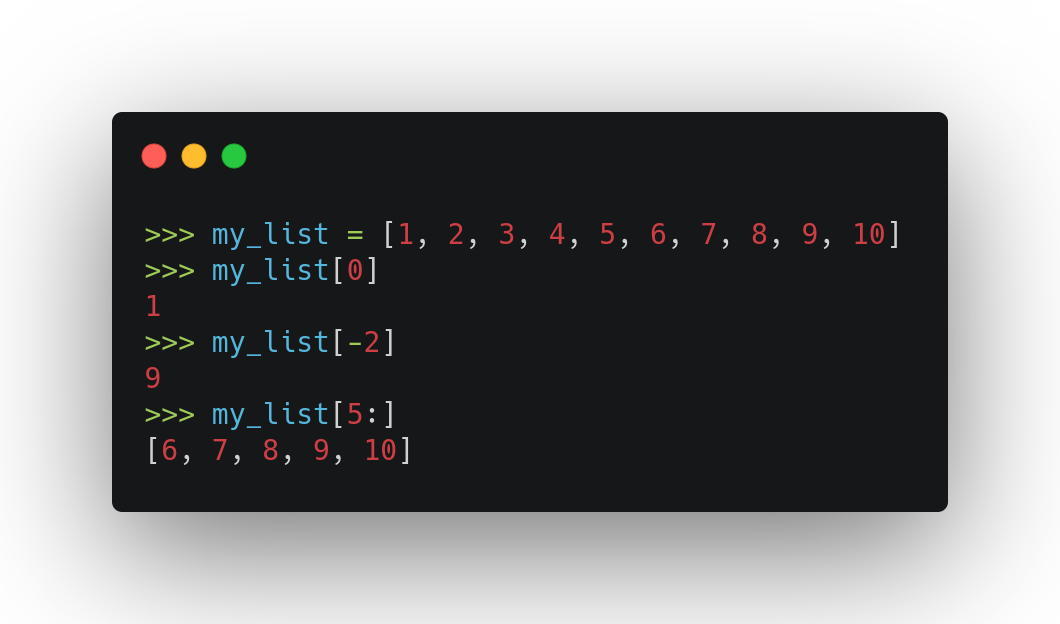
Como é o caso de alguns outros tipos de dados, há mais de uma maneira de criar uma lista: a sintaxe literal [] e a função incorporada list().



As duas linhas de código acima resultam na mesma lista. A diferença é que a função list() leva um iterável(um objeto em que pode ser feito um loop sobre) como argumento e popula a lista resultante com cada um dos valores do iterável. No exemplo acima, "abc" é uma string, que é iterável, e retorna um caractere por vez quando é iterada.

**Acessando os Itens de uma Lista**

Assim como as strings, as listas podem ser indexadas e fatiadas. Isso possível devido à sua característica sequencial.



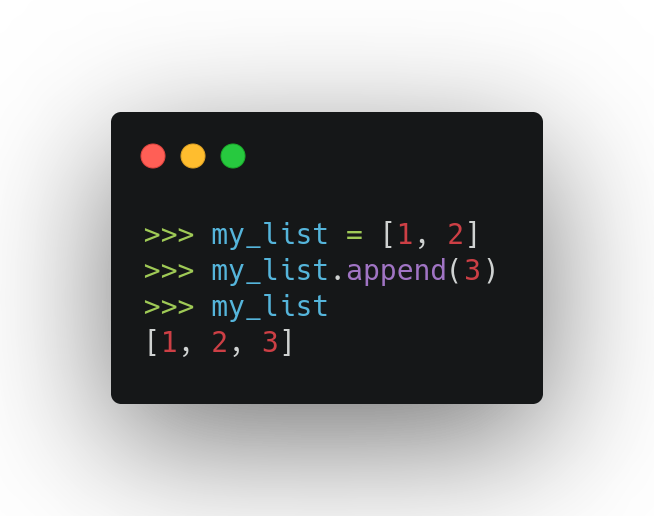
**Removendo os Itens de uma Lista**

Isso mesmo! Há várias maneiras de fazer isso. Você será apresentado a alguns métodos na seção seguinte, mas aqui estão algumas maneiras de fazer isso com palavras-chave Python e fatiamento. A remoção de fatias de uma lista permite que você remova pedaços de uma só vez. Basta atribuir um iterável falso (e.g., "", (), [], {}) e toda a fatia é então removida da lista. É comum usar um literal de string vazia, "". Lembre-se, o fim da fatia não é incluído na própria fatia (ou seja, my\_list[2:5] inclui os índices 2, 3 e 4, mas não o índice 5).

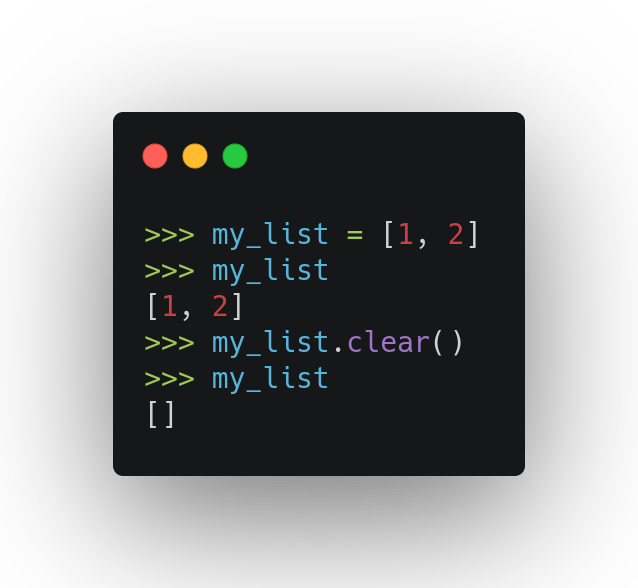
**Métodos da Lista**

Os métodos de lista incorporados permitem que você efetue operações em uma lista. Observe como alguns deles modificam a lista localmente, sem retornar uma nova lista. Isso se deve à característica da lista de ser mutável. Às vezes, isso pode ser uma surpresa se você estiver esperando uma lista como um valor de retorno, já que algumas delas retornam um tipo Nenhum.

* **append()** - adiciona um item ao fim de uma lista



* **clear()** - remove todos os itens de uma lista



* **copy()** - retorna uma cópia de uma lista

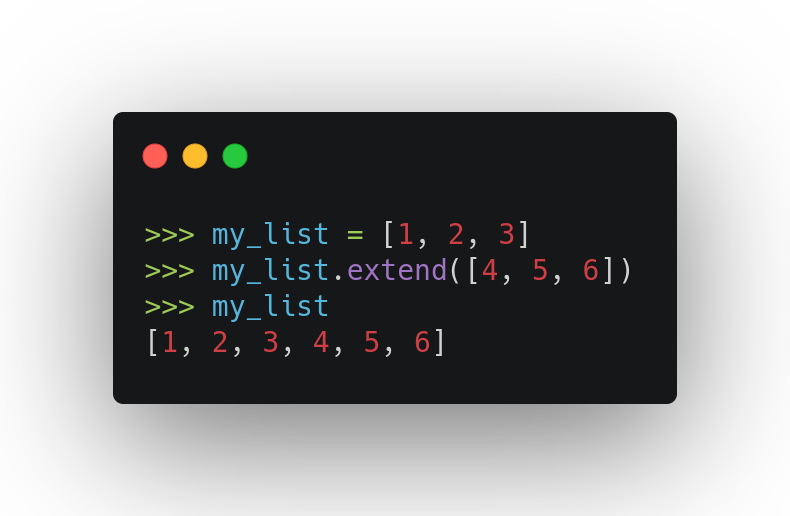


É importante observar que isso produz uma cópia rasa de my\_list1. Isso afetará a forma como as alterações dos itens da lista são tratadas.

* **count()** - conta o número de ocorrências de um valor a partir de uma lista



* **extend()** - adiciona itens de um iterável ao final de uma lista



Pode ser fácil confundir extend() com append(), dado que ambos adicionam algo ao final de uma lista. Tente lembrar-se que o append() adiciona o objeto inteiro à lista…



…mas o extend() adiciona cada item de um iterável.

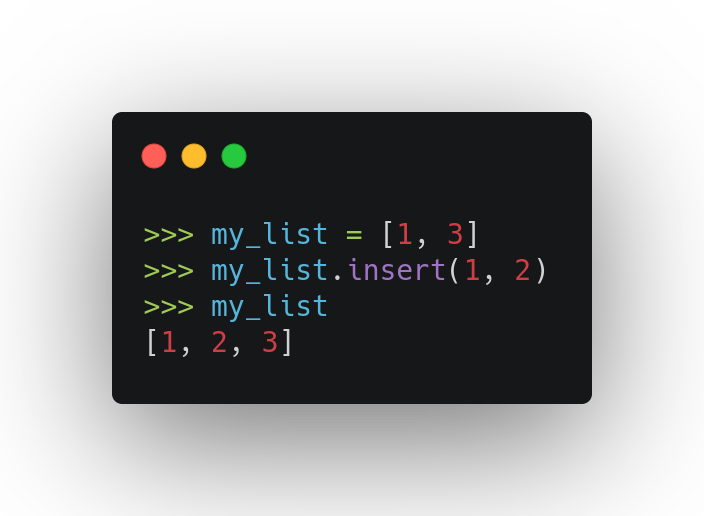


* **index()** - obtém o índice da primeira ocorrência de um valor em uma lista

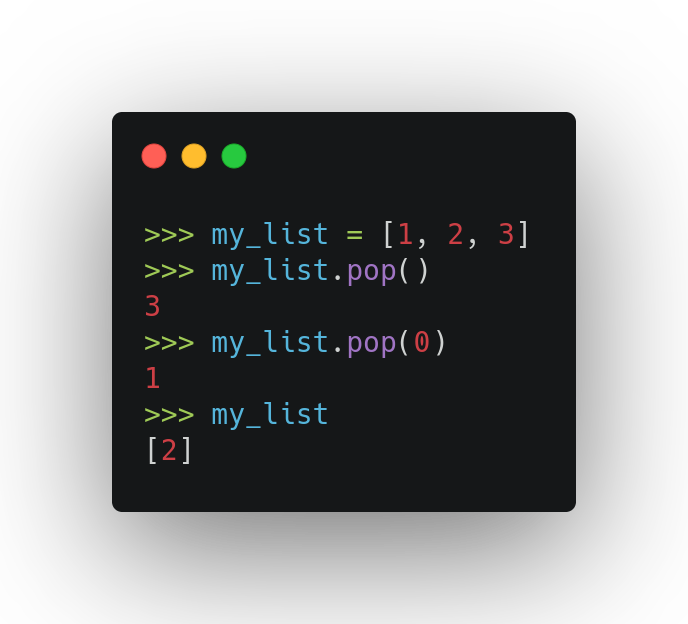


Veja você mesmo o que acontece quando o valor que você está procurando não está na lista. Mais adiante nesse curso, discutiremos como lidar apropriadamente com as exceções.

* **insert()** - adiciona um item em qualquer posição de uma lista

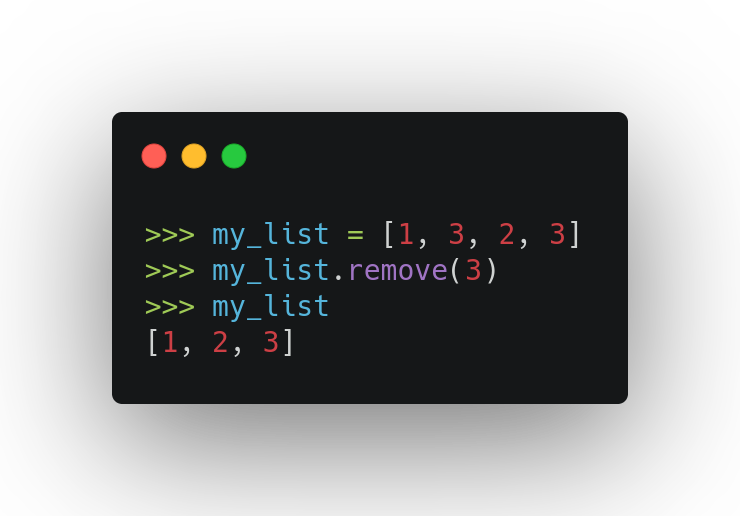


* **pop()** - remove o último(se não for passado parâmetro) e/ou retorna um item de qualquer posição de uma lista(caso o parâmetro seja informado)

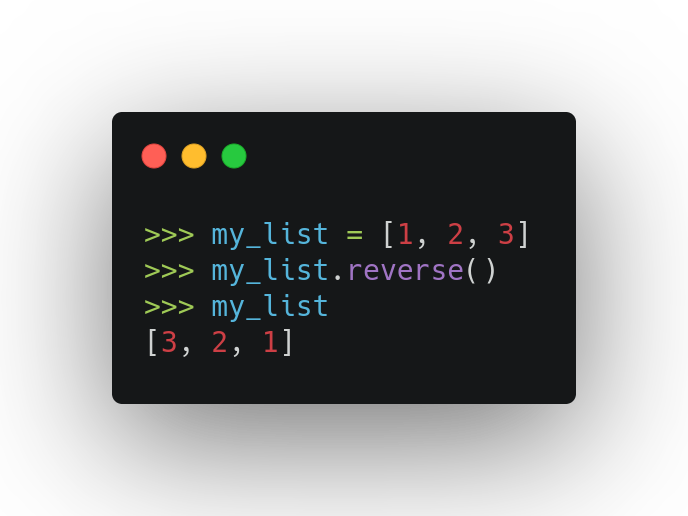


Observe que, sem argumentos, o pop() remove e retorna o último item da lista. A palavra "pop" é parte da terminologia utilizada ao se discutir estruturas de dados como pilhas, o que veremos em breve.

* **remove()** - remove a primeira ocorrência de um valor de uma lista



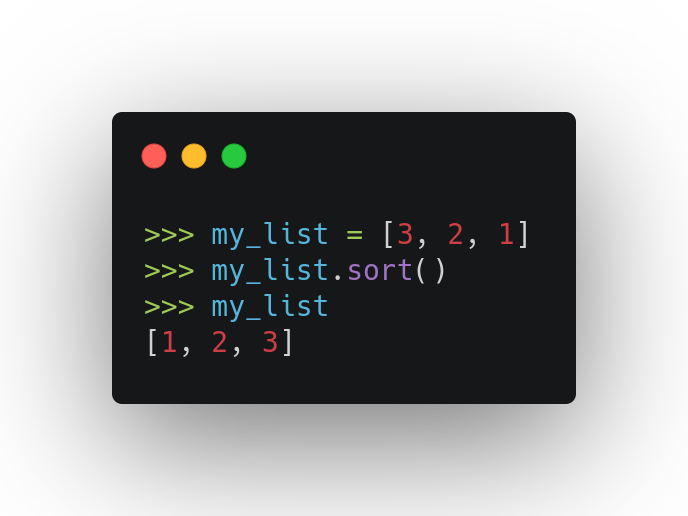
* **reverse()** - inverte a ordem dos itens de uma lista



**Ordenando Listas**

* **sorted()** - ordena os itens de um iterável e os retorna em forma de lista
* **list.sort()** - ordena os itens de uma lista in loco

Os itens da lista podem ser ordenados usando a função incorporada do Python sorted(), ou o método sort() de um objeto de lista. A diferença é que sort() é um método em um objeto de lista que ordena a lista in loco…



…já a função incorporada sorted() pega um iterável e retorna uma lista de seus itens ordenados.



Ambos permitem algum tipo de personalização na forma como a lista é ordenada ao usar um par de argumentos de palavra-chave.

* **key** - uma função que será aplicada a cada item, e terá seu resultado usado como o valor de ordenação



* **reverse** - um indicador booleano para indicar a direção da ordenação

