1. append(): Añade un elemento al final de la lista. Básico:

python

```python
frutas = ['manzana', 'banana']
frutas.append('naranja')
print(frutas)  # ['manzana', 'banana', 'naranja']
```

Avanzado:

python

```python
class ListaAutoordenada:
    def __init__(self):
        self.items = []

    def append(self, item):
        self.items.append(item)
        self.items.sort()

lista = ListaAutoordenada()
lista.append(3)
lista.append(1)
lista.append(2)
print(lista.items)  # [1, 2, 3]
```

2. extend(): Añade los elementos de un iterable al final de la lista. Básico:

python

```python
numeros = [1, 2, 3]
numeros.extend([4, 5])
print(numeros)  # [1, 2, 3, 4, 5]
```

Avanzado:

python

```python
def extender_multiples(lista, *iterables):
    for iterable in iterables:
        lista.extend(iterable)
    return lista

resultado = extender_multiples([1, 2], [3, 4], (5, 6), range(7, 9))
print(resultado)  # [1, 2, 3, 4, 5, 6, 7, 8]
```

3. insert(): Inserta un elemento en una posición específica. Básico:

python

```python
frutas = ['manzana', 'naranja']
frutas.insert(1, 'banana')
print(frutas)  # ['manzana', 'banana', 'naranja']
```

Avanzado:

python

```python
def insertar_ordenado(lista, elemento):
    from bisect import bisect_left
    indice = bisect_left(lista, elemento)
    lista.insert(indice, elemento)
    return lista

numeros = [1, 3, 5, 7]
insertar_ordenado(numeros, 4)
print(numeros)  # [1, 3, 4, 5, 7]
```

4. remove(): Elimina la primera ocurrencia de un elemento. Básico:

python

```python
numeros = [1, 2, 3, 2, 4]
numeros.remove(2)
```

```python
print(numeros)  # [1, 3, 2, 4]
```

Avanzado:

python

```python
def remove_all(lista, elemento):
    return [x for x in lista if x != elemento]

numeros = [1, 2, 3, 2, 4, 2]
numeros = remove_all(numeros, 2)
print(numeros)  # [1, 3, 4]
```

5. pop(): Elimina y devuelve el elemento en una posición específica. Básico:

python

```python
frutas = ['manzana', 'banana', 'naranja']
fruta = frutas.pop(1)
print(fruta)    # banana
print(frutas)   # ['manzana', 'naranja']
```

Avanzado:

python

```python
def pop_multiple(lista, indices):
    indices = sorted(indices, reverse=True)
    return [lista.pop(i) for i in indices]

numeros = [0, 1, 2, 3, 4, 5]
popped = pop_multiple(numeros, [1, 3, 4])
print(popped)   # [4, 3, 1]
print(numeros)  # [0, 2, 5]
```

6. clear(): Elimina todos los elementos de la lista. Básico:

python

```python
numeros = [1, 2, 3]
numeros.clear()
print(numeros)  # []
```

Avanzado:

python

```python
class ListaConHistorial:
    def __init__(self):
        self.items = []
        self.historial = []

    def append(self, item):
        self.items.append(item)

    def clear(self):
        self.historial.append(self.items[:])
        self.items.clear()

lista = ListaConHistorial()
lista.append(1)
lista.append(2)
lista.clear()
lista.append(3)
print(lista.items)      # [3]
print(lista.historial)  # [[1, 2]]
```

7. index(): Devuelve el índice de la primera ocurrencia de un elemento. Básico:

python

```python
frutas = ['manzana', 'banana', 'naranja', 'banana']
print(frutas.index('banana'))  # 1
```

Avanzado:

python

```python
def index_all(lista, elemento):
    return [i for i, x in enumerate(lista) if x == elemento]

numeros = [1, 2, 3, 2, 4, 2, 5]
print(index_all(numeros, 2))  # [1, 3, 5]
```

8. count(): Cuenta el número de ocurrencias de un elemento. Básico:

python

```python
numeros = [1, 2, 2, 3, 2, 4]
print(numeros.count(2))  # 3
```

Avanzado:

python

```python
from collections import Counter

def contar_elementos(lista):
    return Counter(lista)

colores = ['rojo', 'azul', 'verde', 'rojo', 'amarillo', 'azul']
print(contar_elementos(colores))  # Counter({'rojo': 2, 'azul': 2, 'verde': 1, 'amarillo': 1})
```

9. sort(): Ordena la lista in-place. Básico:

python

```python
numeros = [3, 1, 4, 1, 5, 9, 2]
numeros.sort()
print(numeros)  # [1, 1, 2, 3, 4, 5, 9]
```

Avanzado:

python

```python
class Persona:
    def __init__(self, nombre, edad):
        self.nombre = nombre
        self.edad = edad

personas = [Persona("Ana", 30), Persona("Bob", 25), Persona("Carlos", 35)]
personas.sort(key=lambda p: p.edad)
for persona in personas:
    print(f"{persona.nombre}: {persona.edad}")
# Bob: 25
# Ana: 30
# Carlos: 35
```

10. reverse(): Invierte el orden de los elementos de la lista. Básico:

python

```python
numeros = [1, 2, 3, 4, 5]
numeros.reverse()
print(numeros)  # [5, 4, 3, 2, 1]
```

Avanzado:

python

```python
def reverse_groups(lista, n):
    return [x for group in zip(*[iter(lista)]*n) for x in reversed(group)]

numeros = list(range(1, 11))
print(reverse_groups(numeros, 3))  # [3, 2, 1, 6, 5, 4, 9, 8, 7, 10]
```

11. copy(): Devuelve una copia superficial de la lista. Básico:

python

```python
original = [1, 2, 3]
copia = original.copy()
```

```python
copia.append(4)
print(original)  # [1, 2, 3]
print(copia)     # [1, 2, 3, 4]
```

Avanzado:

python

```python
import copy

def deep_copy_list(lista):
    return copy.deepcopy(lista)

original = [[1, 2], [3, 4]]
copia_profunda = deep_copy_list(original)
copia_profunda[0][0] = 5
print(original)       # [[1, 2], [3, 4]]
print(copia_profunda) # [[5, 2], [3, 4]]
```

12. filter(): Crea un iterador a partir de elementos que cumplen una condición. Básico:

python

```python
numeros = [1, 2, 3, 4, 5, 6]
pares = list(filter(lambda x: x % 2 == 0, numeros))
print(pares)  # [2, 4, 6]
```

Avanzado:

python

```python
def filtrar_por_atributos(objetos, **condiciones):
    def cumple_condiciones(obj):
        return all(getattr(obj, attr) == valor for attr, valor in condiciones.items())
    return list(filter(cumple_condiciones, objetos))

class Producto:
    def __init__(self, nombre, precio, categoria):
```

```python
        self.nombre = nombre
        self.precio = precio
        self.categoria = categoria

productos = [
    Producto("Manzana", 0.5, "Fruta"),
    Producto("Leche", 2.0, "Lácteo"),
    Producto("Pan", 1.5, "Panadería"),
    Producto("Queso", 3.0, "Lácteo")
]

lacteos = filtrar_por_atributos(productos, categoria="Lácteo")
for producto in lacteos:
    print(f"{producto.nombre}: ${producto.precio}")
# Leche: $2.0
# Queso: $3.0
```

13. map(): Aplica una función a cada elemento de la lista. Básico:

python

```python
numeros = [1, 2, 3, 4, 5]
cuadrados = list(map(lambda x: x**2, numeros))
print(cuadrados)  # [1, 4, 9, 16, 25]
```

Avanzado:

python

```python
def aplicar_operaciones(valor, *operaciones):
    for operacion in operaciones:
        valor = operacion(valor)
    return valor

numeros = [1, 2, 3, 4, 5]
operaciones = [
    lambda x: x * 2,
    lambda x: x + 3,
```

```
    lambda x: x ** 2
]
resultados = list(map(lambda x: aplicar_operaciones(x, *operaciones), numeros))
print(resultados)  # [25, 49, 81, 121, 169]
```

14. reduce(): Aplica una función de dos argumentos acumulativamente a los elementos de la lista. Básico:

python

```
from functools import reduce
numeros = [1, 2, 3, 4, 5]
suma = reduce(lambda x, y: x + y, numeros)
print(suma)  # 15
```

Avanzado:

python

```
def combinar_diccionarios(dict1, dict2):
    return {k: dict1.get(k, 0) + dict2.get(k, 0) for k in set(dict1) | set(dict2)}

ventas_por_dia = [
    {"manzanas": 3, "peras": 2},
    {"manzanas": 2, "naranjas": 3},
    {"peras": 1, "naranjas": 2}
]

total_ventas = reduce(combinar_diccionarios, ventas_por_dia)
print(total_ventas)  # {'manzanas': 5, 'peras': 3, 'naranjas': 5}
```