# Métodos útiles para utilizar listas en Python

**Resumen**

El informe presenta una revisión de los métodos más importantes para el uso de listas, incluyendo la sobrecarga de operadores y los métodos de ordenamiento. Se realizó una búsqueda utilizando buscadores académicos y el sitio oficial de Python para recopilar la información.

**Palabras clave**

Python, listas, estructuras de datos, sobrecarga de operadores, programación

## Introducción

Python es un lenguaje de programación interpretado, de alto nivel, popular debido a su facilidad de uso, sintaxis sencilla, legibilidad del código y gran cantidad de bibliotecas y herramientas disponibles.

El informe de investigación tiene como objetivo  los métodos más destacados para el manejo de listas en Python, enfocándose en aquellos que son de utilidad. Estos métodos facilitan la manipulación de los elementos en las listas, permiten la creación de programas más eficientes y optimizados.

## 1.Desarrollo

Las listas son una estructura de datos muy utilizada debido a su versatilidad y facilidad de uso. Son objetos mutables que pueden contener una secuencia de elementos, y que permiten acceder, modificar, agregar o eliminar elementos según sea necesario.

Se presentarán algunos de los métodos más destacados para el manejo de listas.

**1 Método append():** El método append() se utiliza para agregar un elemento al final de una lista. Este método toma un argumento que representa el elemento que se va a agregar.

```
animal1 = Animal("gato", "mamífero")
animal2 = Animal("perro", "mamífero")
animales = [animal1, animal2]

otro_animal = Animal("loro", "ave")
animales.append(otro_animal)
```

**1.2 extend():**  El método extend() se utiliza para agregar varios elementos al final de una lista. Este método toma un iterable como argumento (por ejemplo, otra lista) y agrega cada elemento individualmente al final de la lista original.

```
animales1 = [animal1, animal2]
animales2 = [Animal("pato", "ave"), Animal("pez", "acuático")]
animales1.extend(animales2)
```

**1.3 insert():** permite insertar un elemento en una posición específica de la lista. Por ejemplo:

```
animales = [animal1, animal2]
animales.insert(1, Animal("ratón", "mamífero"))
```

**1.4 remove():** Permite remover el primer elemento de la lista que sea igual al elemento pasado como argumento.

Ejemplificando:

```
animales = [animal1, animal2, otro_animal]
animales.remove(animal1)
```

***1.5 pop():*** *Permite remover y devolver el último elemento de una lista. Si se le pasa como argumento un índice, removerá y devolverá el elemento correspondiente a dicho índice.*

```
animales = [animal1, animal2, otro_animal]
elemento = animales.pop()
```

***Métodos de ordenamiento de listas***

***2.1 sort()*** *permite ordenar una lista en orden ascendente o descendente. También es posible personalizar el ordenamiento mediante una función de comparación. Por ejemplo:*

```
animales = [animal2, animal1, otro_animal]
animales.sort(key=lambda animal: animal.nombre)
```

***2.2 reverse()*** *permite invertir el orden de los elementos en una lista. Por ejemplo:*

```
animales = [animal1, animal2, otro_animal]
animales.reverse()
```

***2.3 sorted():*** *se utiliza para ordenar los elementos de una lista en orden ascendente o descendente y devuelve una nueva lista ordenada. Este método no modifica la lista original.*

```
class Coche:
def __init__(self, marca, modelo):
self.marca = marca
self.modelo = modelo

coches = [Coche("Ford", "Fiesta"), Coche("Toyota", "Corolla"), Coche("Honda",
"Civic")]
coches_ordenados = sorted(coches, reverse=True, key=lambda x: x.modelo)

for coche in coches_ordenados:
print(coche.modelo)
```

**Sobrecarga de operadores**

La sobrecarga de operadores es una característica de la programación orientada a objetos que permite a los programadores definir la funcionalidad de los operadores para sus propias clases.

**add():** operador + se puede sobrecargar con el método add() para permitir la concatenación de objetos.

**eq():** operador == se puede sobrecargar con el método eq() para permitir la comparación de objetos

**lt():** operador < se puede sobrecargar con el método lt() para permitir la comparación de objetos.

En el siguiente ejemplo, la sobrecarga de append(), el operador + y otros métodos nos permiten manipular objetos de la clase ListaDeAnimales de una manera más intuitiva y natural, como si estuviéramos trabajando con una lista de Python estándar.

```
class Animal:
    def __init__(self, nombre, tipo):
        self.nombre = nombre
        self.tipo = tipo

class ListaDeAnimales:
    def __init__(self, animales=None):
        if animales is None:
            self.animales = []
        else:
            self.animales = animales
        self.cantidad_animales = len(self.animales)

    def __add__(self, otra_lista):
        nueva_lista = ListaDeAnimales(self.animales + otra_lista.animales)
        nueva_lista.cantidad_animales = len(nueva_lista.animales)
        return nueva_lista

    def __len__(self):
        return self.cantidad_animales

    def __getitem__(self, indice):
        return self.animales[indice]

    def __setitem__(self, indice, nuevo_animal):
        self.animales[indice] = nuevo_animal

    def __delitem__(self, indice):
        del self.animales[indice]
        self.cantidad_animales -= 1

    def __iter__(self):
        return iter(self.animales)

    def append(self, nuevo_animal):
        self.animales.append(nuevo_animal)
        self.cantidad_animales += 1
```

```
lista1 = ListaDeAnimales([Animal("gato", "mamífero"), Animal("perro",
"mamífero")])
lista2 = ListaDeAnimales([Animal("loro", "ave")])

lista3 = lista1 + lista2
print(len(lista3))  # salida: 3

lista3.append(Animal("tigre", "mamífero"))
print(len(lista3))  # salida: 4
```

## Conclusión

Python es un lenguaje de programación muy utilizado y popular debido a su sintaxis sencilla, legibilidad del código, facilidad de uso y gran cantidad de bibliotecas y herramientas disponibles. Las listas son una estructura de datos muy útil y versátil en Python, y existen varios métodos importantes para su manipulación, como append(), extend(), insert(), remove() y sort(). Además, la sobrecarga de operadores en Python permite a los programadores definir la funcionalidad de los operadores para sus propias clases, lo que facilita la manipulación de objetos de una manera más intuitiva y natural.

## Bibliografía

Müller, A. C., & Guido, S. (2016). Introduction to machine learning with Python: A guide for data scientists. O'Reilly Media, Inc.

Sweigart, A. (2017). Automate the boring stuff with Python: Practical programming for total beginners. No Starch Press.

Documentación oficial de Python: https://docs.python.org/3/
Python Software Foundation: https://www.python.org/
GeeksforGeeks: https://www.geeksforgeeks.org/