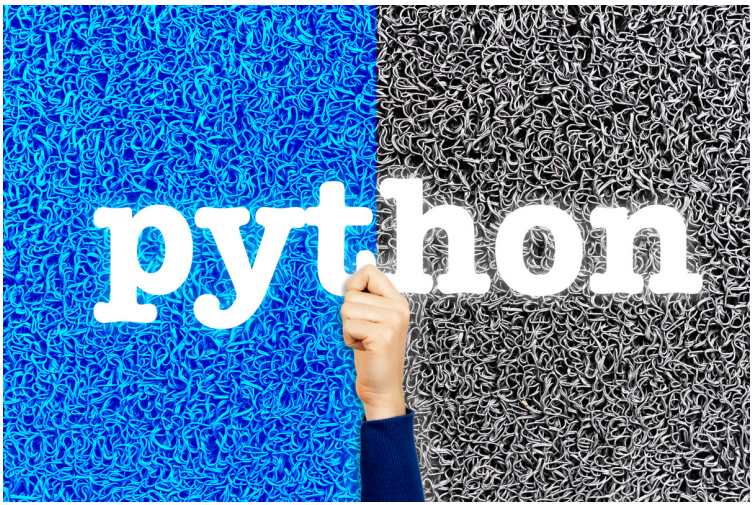

LENGUAJES DE
PROGRAMACIÓN
PYTHON

Algoritmos de búsqueda y ordenación.

En el campo de la programación, la capacidad de buscar información y ordenar datos de manera eficiente es esencial. Python, como lenguaje de programación versátil, ofrece una variedad de algoritmos de búsqueda y ordenación que nos permiten realizar estas tareas de manera eficaz. En esta lección, exploraremos algunos de los algoritmos más comunes utilizados para buscar y ordenar datos en Python. A través de ejemplos y demostraciones, aprenderemos cómo implementar y utilizar estos algoritmos en nuestros programas.



Algoritmos de Búsqueda

Los algoritmos de búsqueda nos permiten encontrar un elemento específico en una colección de datos. A continuación, se presentan dos algoritmos populares de búsqueda en Python:

Búsqueda Lineal

La búsqueda lineal es un enfoque simple pero efectivo para buscar un elemento en una lista. Consiste en recorrer la lista elemento por elemento hasta encontrar el valor buscado. Veamos un ejemplo:

```
def busqueda_lineal(lista, valor):
    for elemento in lista:
        if elemento == valor:
            return True
    return False

# Ejemplo de uso
mi_lista = [4, 2, 6, 1, 8, 3]
resultado = busqueda_lineal(mi_lista, 6)
```

En el ejemplo anterior, implementamos una función llamada busqueda_lineal que toma una lista y un valor como parámetros. La función recorre la lista y compara cada elemento con el valor buscado. Si encuentra una coincidencia, retorna True. Si termina de recorrer la lista sin encontrar el valor, retorna False.

Búsqueda Binaria

La búsqueda binaria es un algoritmo eficiente para buscar elementos en una lista ordenada. Funciona dividiendo repetidamente la lista a la mitad y descartando la mitad en la que no puede estar el valor buscado. Veamos un ejemplo:

```
def busqueda_binaria(lista, valor):
    inicio = 0
    fin = len(lista) - 1
    while inicio <= fin:
        medio = (inicio + fin) // 2
        if lista[medio] == valor:
            return True
        elif lista[medio] < valor:
            inicio = medio + 1
        else:
            fin = medio - 1
    return False

# Ejemplo de uso
mi_lista = [1, 2, 3, 4, 6, 8]
resultado = busqueda_binaria(mi_lista, 6)
```

En el ejemplo anterior, implementamos una función llamada busqueda_binaria que toma una lista ordenada y un valor como parámetros.

La función inicializa dos variables, inicio y fin, que representan los índices del rango actual de búsqueda. Luego, utiliza un bucle while para dividir repetidamente la lista a la mitad y ajustar los límites de búsqueda según corresponda.

Si encuentra el valor buscado, retorna True. Si el rango de búsqueda se reduce a cero y el valor no se encuentra, retorna False.

Algoritmos de Ordenación

Los algoritmos de ordenación nos permiten organizar los elementos de una lista en un orden específico, como ascendente o descendente. A continuación, se presentan dos algoritmos populares de ordenación en Python:

Ordenación Burbuja

La ordenación burbuja es un algoritmo simple pero ineficiente que compara y intercambia repetidamente los pares de elementos adyacentes hasta que la lista esté ordenada. Veamos un ejemplo:

```
def ordenacion_burbuja(lista):
    n = len(lista)
    for i in range(n - 1):
        for j in range(n - 1 - i):
            if lista[j] > lista[j + 1]:
                lista[j], lista[j + 1] =
lista[j + 1], lista[j]

# Ejemplo de uso
mi_lista = [4, 2, 6, 1, 8, 3]
ordenacion_burbuja(mi_lista)
```

En el ejemplo anterior, implementamos una función llamada ordenacion_burbuja que toma una lista como parámetro. La función utiliza dos bucles for anidados para comparar y intercambiar los pares de elementos adyacentes. El bucle exterior se repite n-1 veces, donde n es la longitud de la lista. El bucle interior se repite n-1-i veces en cada iteración del bucle exterior, donde i representa el número de elementos que ya están en su posición final.

Ordenación por Inserción

La ordenación por inserción es un algoritmo eficiente que construye una lista ordenada insertando elementos uno a uno en su posición correcta. Veamos un ejemplo:

```
def ordenacion_insercion(lista):
    n = len(lista)
    for i in range(1, n):
        valor_actual = lista[i]
        j = i - 1
                while  j  >=  0  and  lista[j]  >
valor_actual:
            lista[j + 1] = lista[j]
            j -= 1
        lista[j + 1] = valor_actual

# Ejemplo de uso
mi_lista = [4, 2, 6, 1, 8, 3]
ordenacion_insercion(mi_lista)
```

En el ejemplo anterior, implementamos una función llamada ordenacion_insercion que toma una lista como parámetro. La función utiliza un bucle for para iterar sobre los elementos de la lista desde la posición 1 hasta la posición n-1. En cada iteración, guarda el valor actual y compara los elementos anteriores en orden descendente. Si encuentra un elemento mayor, desplaza ese elemento a la derecha y continúa hacia atrás hasta encontrar su posición correcta. Finalmente, inserta el valor actual en su posición correcta.

Los algoritmos de búsqueda y ordenación son herramientas fundamentales para organizar y buscar información eficientemente en Python. Los algoritmos de búsqueda, como la búsqueda lineal y la búsqueda binaria, nos permiten encontrar elementos específicos en una colección de datos. Por otro lado, los algoritmos de ordenación, como la ordenación burbuja y la ordenación por inserción, nos ayudan a organizar los elementos de una lista en un orden deseado. La elección del algoritmo adecuado dependerá del tamaño de los datos y de la eficiencia deseada.