**LAPORAN PRAKTIKUM**

**STRUKTUR DATA**

**MODUL KE-07**

**JUDUL PRAKTIKUM**



**Disusun Oleh:**

**Nama :** Restu Wibisono

**NPM :** 2340506061

**Kelas :** 03 (Tiga)

**Program Studi S1 Teknologi Informasi**

**Fakultas Teknik, Universitas Tidar**

**Genap 2023/2024**

1. **Tujuan Praktikum**

Adapun tujuan praktikum ini sebagai berikut :

1. Mahasiswa mampu menerapkan konsep tree pada bahasa pemrograman python
2. **Dasar Teori**

Struktur data pohon adalah sebuah struktur hierarkis yang digunakan untuk mewakili dan mengatur data dengan cara yang mudah untuk dinavigasi dan dicari. Ini merupakan kumpulan dari node yang saling terhubung melalui tepi dan memiliki hubungan hierarkis di antara node-node tersebut. Terminologi dalam Struktur Data Pohon:

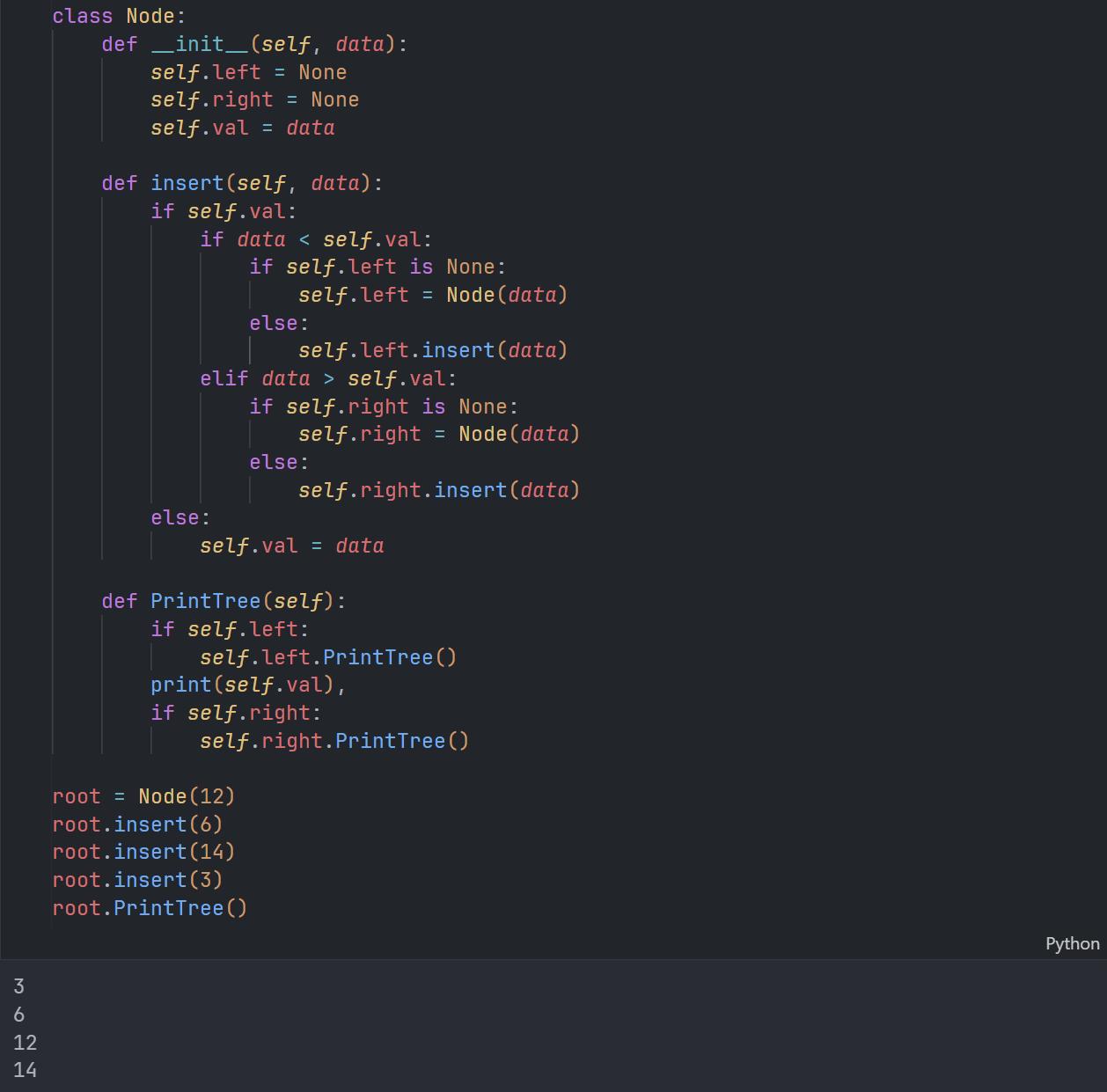
* Node Induk: Node yang menjadi pendahulu suatu node disebut node induk dari node tersebut. {B} adalah node induk dari {D, E}.
* Node Anak: Node yang merupakan pengganti langsung dari suatu node disebut node anak dari node tersebut. Contoh: {D, E} adalah node anak dari {B}.
* Node Akar: Node paling atas dari sebuah pohon atau node yang tidak memiliki node induk disebut node akar. {A} adalah node akar dari pohon tersebut. Sebuah pohon yang tidak kosong harus mengandung tepat satu node akar dan tepat satu jalur dari akar ke semua node lain dalam pohon.
* Node Daun atau Node Eksternal: Node yang tidak memiliki node anak disebut node daun. {K, L, M, N, O, P, G} adalah node daun dari pohon tersebut.
* Silsilah dari Sebuah Node: Node-node pendahulu di jalur dari akar ke node tersebut disebut silsilah dari node tersebut. {A, B} adalah node silsilah dari node {E}.
* Keturunan: Sebuah node x adalah keturunan dari node lain y jika dan hanya jika y adalah leluhur dari x.
* Saudara: Anak-anak dari node induk yang sama disebut saudara. {D, E} disebut saudara.
* Tingkat sebuah node: Jumlah tepi pada jalur dari node akar ke node tersebut. Node akar memiliki tingkat 0.
* Node Internal: Sebuah node dengan setidaknya satu node anak disebut Node Internal.
* Tetangga dari Sebuah Node: Node induk atau anak dari node tersebut disebut tetangga dari node tersebut.
* Subpohon: Setiap node dari pohon bersama dengan keturunannya.

1. **Hasil dan Pembahasan**
   * + 1. Implementasi Sederhana dari Binary Tree



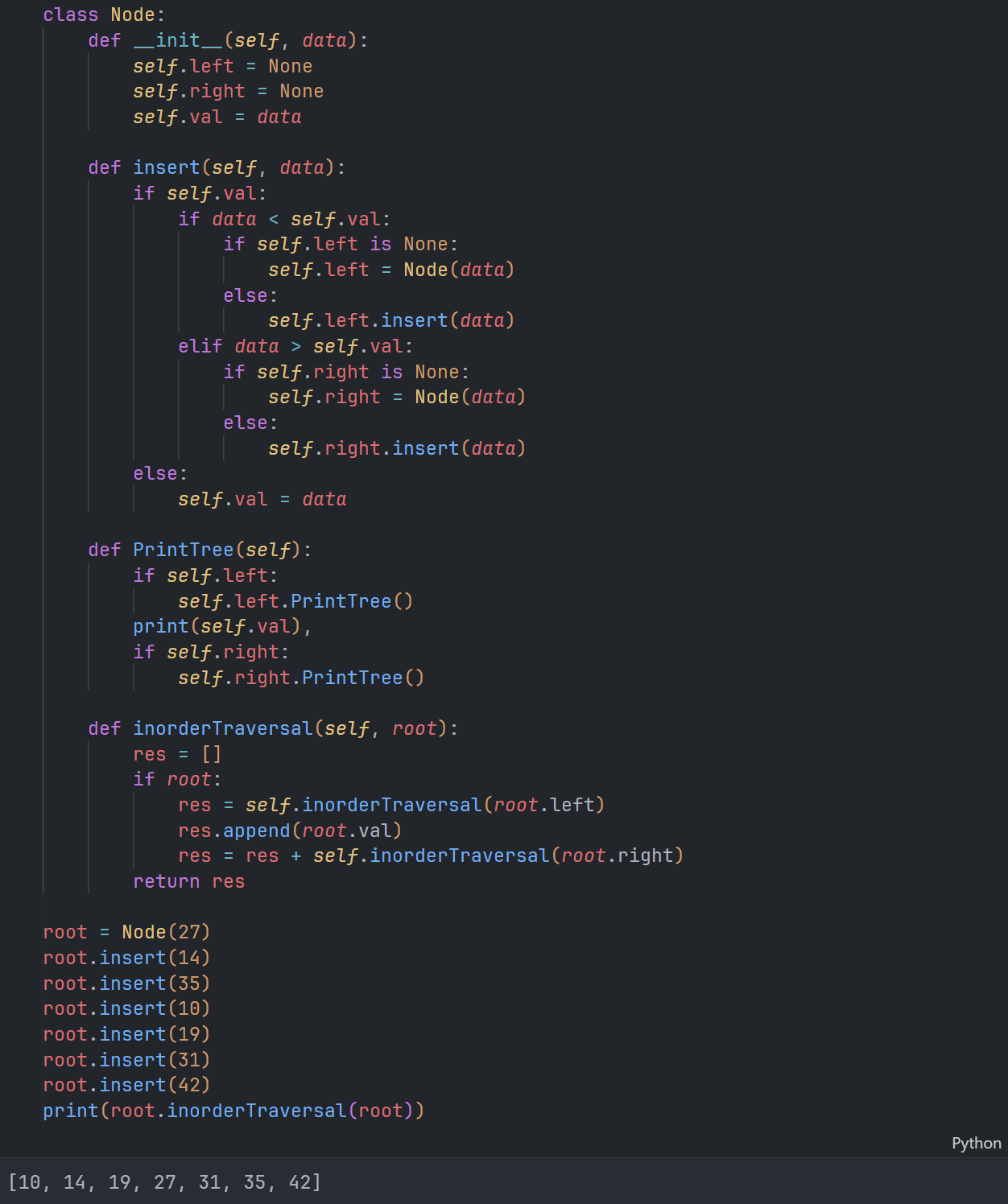
(Gambar 3.1.1)

* 1. Membuat kelas untuk Node (simpul) dalam pohon biner. Yang akan berfungsi menjadi cetakan untuk membuat objek simpul dalam pohon.
  2. metode konstruktor dengan def \_\_init\_\_(self, key): untuk kelas ‘Node’. Saat objek Node dibuat, ini akan dipanggil yang berfungsi menginisialisasi objek. Dengan menerima parameter ‘key’, yang merupakan nilai yang akan disimpan di dalam simpul.
  3. ‘self.left’ berfungsi untuk menyimpan referensi ke simpul anak kiri. Pembuatan ‘Node’, atribut ini diatur ke ‘None’, yang menunjukkan simpul tersebut belum memiliki anak sampul.
  4. Sama seperti sebelumnya, ‘self.right’ berfungsi untuk menyimpan referensi ke simpul anak kanan. Pada pembuatan objek ‘Node’, atribut ini diatur ke ‘None’, untuk menunjukkan simpul tersebut awalnya tidak memiliki anak kanan.
  5. self.val = key berfungsi menyimpan nilai dari simpul itu sendiri. Nilai ini diambil dari parameter ‘key’ yang diterima oleh konstruktor.
  6. Idiom if \_\_name\_\_ == '\_\_main\_\_': Python yang digunakan untuk beberapa kode, bukan jika diimpor sebagai modul.
  7. Pada root = Node(1”-5”) berfungsi membuat objek ‘Node’ dengan nilai kunci 1 dan menyimpannya ke dalam variabel ‘root’, yang akan menjadi akar dari pohon biner.Penyisipan Elemen



(Gambar 3.1.2)

* 1. Untuk mendefinisikan kelas Node digunakan ‘class Node’ merepresentasikan simpul dalam BST.
  2. Konstruktor menggunakan def \_\_init\_\_(self, data):. Ketika Node dibuat, metode akan dipanggil untuk menginisialisasi. Ini akan menerima parameter ‘data’, yang merupakan nilai yang akan disimpan dalam simpul.
  3. ‘self.left = None’ dan ‘self.right = None’: adalah atribut ‘left’ dan ‘right’ yang digunakan untuk menyimpan referensi dari simpul saat ini. Jika simpul baru di buat, kedua atribut di atur menjadi ‘None’ .
  4. Atribut ‘val’ berfungsi untuk menyimpan nilai dari simpul itu sendiri.
  5. Metode ‘insert’ berfungsi untuk menyisipkan elemen baru. Ini menerima parameter ‘data’, yang mana merupakan nilai dari elemen yang akan di sisipkan.
  6. ‘if data < self.val:’ dan ‘if data > self.val:’: berfungsi untuk membandingkan nilai ‘data’ dengan nilai ‘self.val’.
  7. ‘if self.left is None:’ dan ‘if self.right is None:’: berfungsi untuk memeriksa anak kiri atau anak kanan dari simpul saat ini kosong. Jika kosong maka elemen baru akan.
  8. ‘self.left.insert(data)’ dan ‘self.right.insert(data)’: Jika anak kiri atau anak kanan tidak kosong, metode ‘insert’ akan dipanggil rekursif.
  9. ‘def PrintTree(self):’ Metode ‘PrintTree’ digunakan untuk mencetak nilai-nilai dalam BST secara inorder traversal.
  10. ‘print(self.val)’ berfungsi untuk mencetak nilai dari simpul saat ini.
      + 1. Lintasan Elemen In-order Traversal

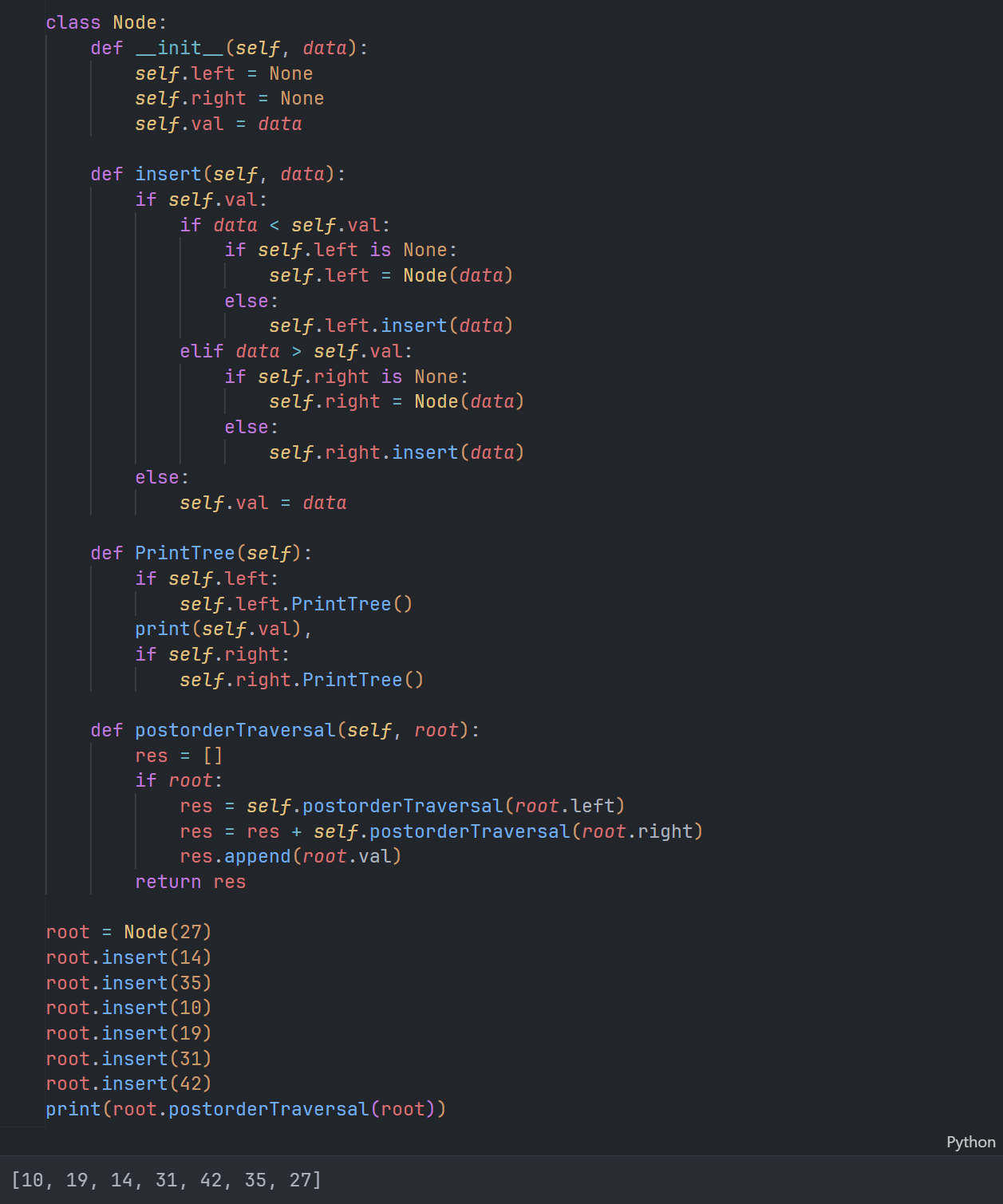


(Gambar 3.2.1)

1. Untuk mendefinisikan kelas Node digunakan ‘class Node:’ merepresentasikan simpul dalam pohon biner.
2. Konstruktor menggunakan def \_\_init\_\_(self, data):. Ketika Node dibuat, metode akan dipanggil untuk menginisialisasi. Ini akan menerima parameter ‘data’, yang merupakan nilai yang akan disimpan dalam simpul.
3. ‘self.left = None’ dan ‘self.right = None’: adalah atribut ‘left’ dan ‘right’ yang digunakan untuk menyimpan referensi dari simpul saat ini. Jika simpul baru di buat, kedua atribut di atur menjadi ‘None’ .
4. Atribut ‘val’ berfungsi untuk menyimpan nilai dari simpul itu sendiri.
5. Metode ‘insert’ berfungsi untuk menyisipkan elemen baru. Ini menerima parameter ‘data’, yang mana merupakan nilai dari elemen yang akan di sisipkan.
6. Metode ‘PrintTree’ digunakan untuk mencetak nilai dalam pohon biner secara inorder.
7. Pada ‘inorderTraversal’ Berfungsi untuk melakukan penelusuran dari pohon biner yang akan mengembalikan daftar nilai-nilai simpul urutan inorder.
8. Variabel ‘res’ digunakan untuk menyimpan hasil penelusuran inorder.
9. Selanjutnya memeriksa apakah ‘root’ bukan ‘None’. Jika tidak, sampul akan melakukan penelusuran.
10. Lalu melakukan rekursi yang berfungsi menelusuri anak kiri dari ‘root’ dan menyimpan hasilnya dalam ‘res’.
11. Menambahkan nilai ‘root’ ke dalam ‘res’ yang dilakukan setelah menelusuri anak kiri, simpul yang diproses terakhir adalah simpul yang paling kiri.
12. ‘return res’ akan mengembalikan daftar yang berisi nilai dalam inorder.
13. ‘root = Node(27)’ Membuat objek ‘root’ dengan nilai 27.
14. Menyisipkan beberapa nilai ke dalam pohon biner menggunakan metode ‘insert’.
15. Mencetak hasil penelusuran inorder dengan ‘print(root.inorderTraversal(root))’ dari pohon biner dengan menggunakan metode ‘inorderTraversal’.
    * + 1. Lintasan Elemen Pre-order Traversal

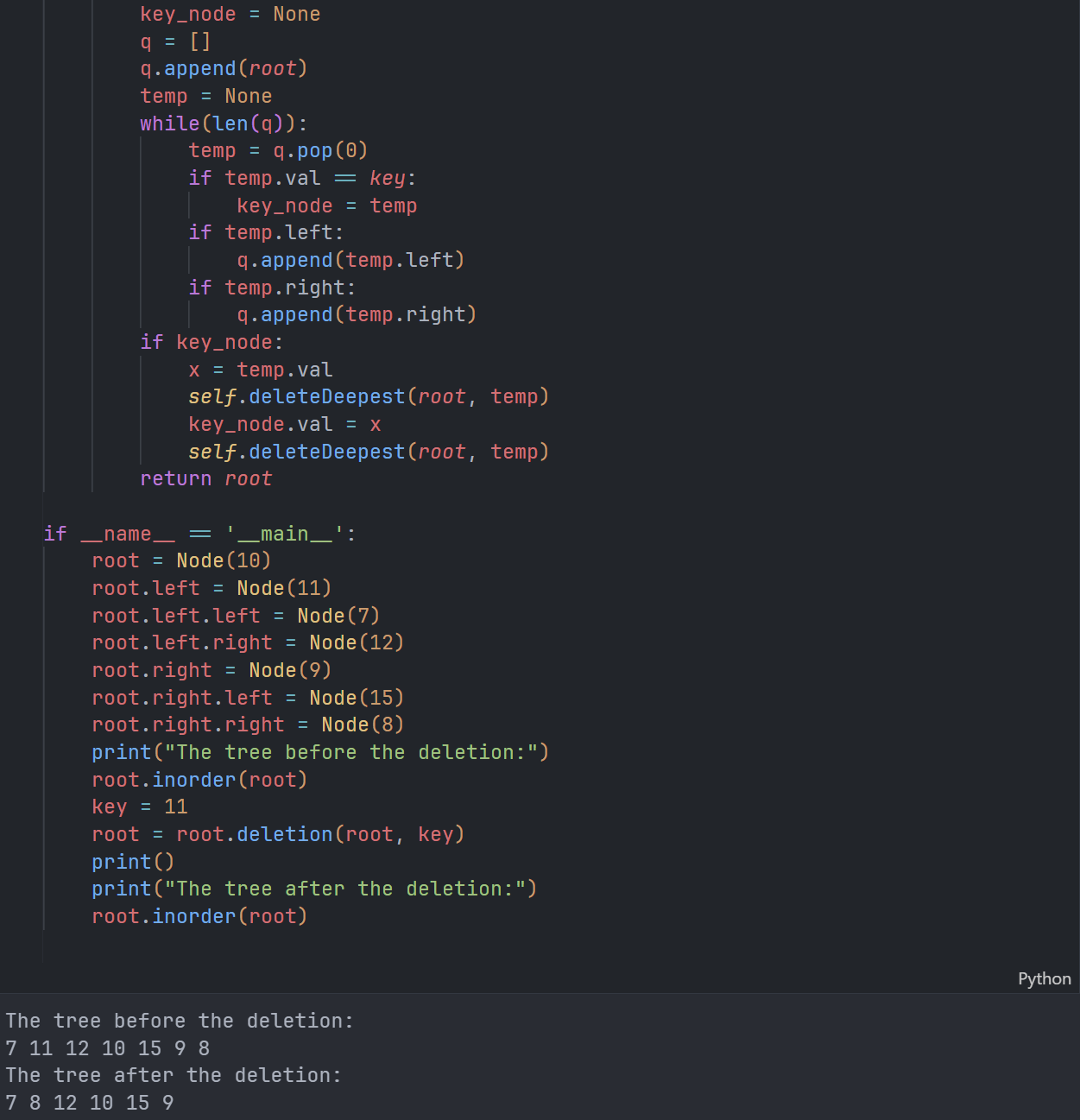


1. ‘class Node:’: Mendefinisikan kelas Node untuk merepresentasikan simpul dalam pohon biner.
2. metode konstruktor dengan def \_\_init\_\_(self, key): untuk kelas ‘Node’. Saat objek Node dibuat, ini akan dipanggil yang berfungsi menginisialisasi objek. Dengan menerima parameter ‘key’, yang merupakan nilai yang akan disimpan di dalam simpul.
3. ‘self.left’ berfungsi untuk menyimpan referensi ke simpul anak kiri. Pembuatan ‘Node’, atribut ini diatur ke ‘None’, yang menunjukkan simpul tersebut belum memiliki anak sampul.
4. self.val = key berfungsi menyimpan nilai dari simpul itu sendiri. Nilai ini diambil dari parameter ‘key’ yang diterima oleh konstruktor.
5. Metode ‘insert’ berfungsi untuk menyisipkan elemen baru. Ini menerima parameter ‘data’, yang mana merupakan nilai dari elemen yang akan di sisipkan.
6. ‘def PrintTree(self):’: Metode ‘PrintTree’ digunakan untuk mencetak nilai-nilai dalam pohon biner secara inorder.
7. ‘def preorderTraversal(self, root):’: Metode ‘preorderTraversal’ digunakan untuk melakukan penelusuran pre-order dari pohon biner. Ini akan mengembalikan daftar yang berisi nilai-nilai simpul dalam urutan pre-order.
8. ‘res = []’: Variabel ‘res’ digunakan untuk menyimpan hasil penelusuran pre-order.
9. ‘if root:’: Memeriksa apakah ‘root’ bukan ‘None’. Jika tidak, artinya masih ada simpul yang harus ditelusuri.
10. ‘res.append(root.val)’: Menambahkan nilai ‘root’ ke dalam ‘res’. Ini dilakukan sebelum melakukan penelusuran rekursif ke anak kiri dan anak kanan, karena pada penelusuran pre-order, simpul saat ini diproses terlebih dahulu.
11. ‘res = res + self.preorderTraversal(root.left)’: Melakukan rekursi untuk menelusuri anak kiri dari ‘root’ dan menambahkan hasilnya ke dalam ‘res’.
12. ‘res = res + self.preorderTraversal(root.right)’: Melakukan rekursi untuk menelusuri anak kanan dari ‘root’ dan menambahkan hasilnya ke dalam ‘res’.
13. ‘return res’: Mengembalikan daftar yang berisi nilai-nilai dalam urutan pre-order.
14. ‘root = Node(27)’: Membuat objek ‘root’ dengan nilai 27.
15. Menyisipkan beberapa nilai ke dalam pohon biner menggunakan metode ‘insert’.
16. ‘print(root.preorderTraversal(root))’: Mencetak hasil penelusuran pre-order dari pohon biner dengan menggunakan metode ‘preorderTraversal’.
    * + 1. Lintasan Elemen Post-order Traversal



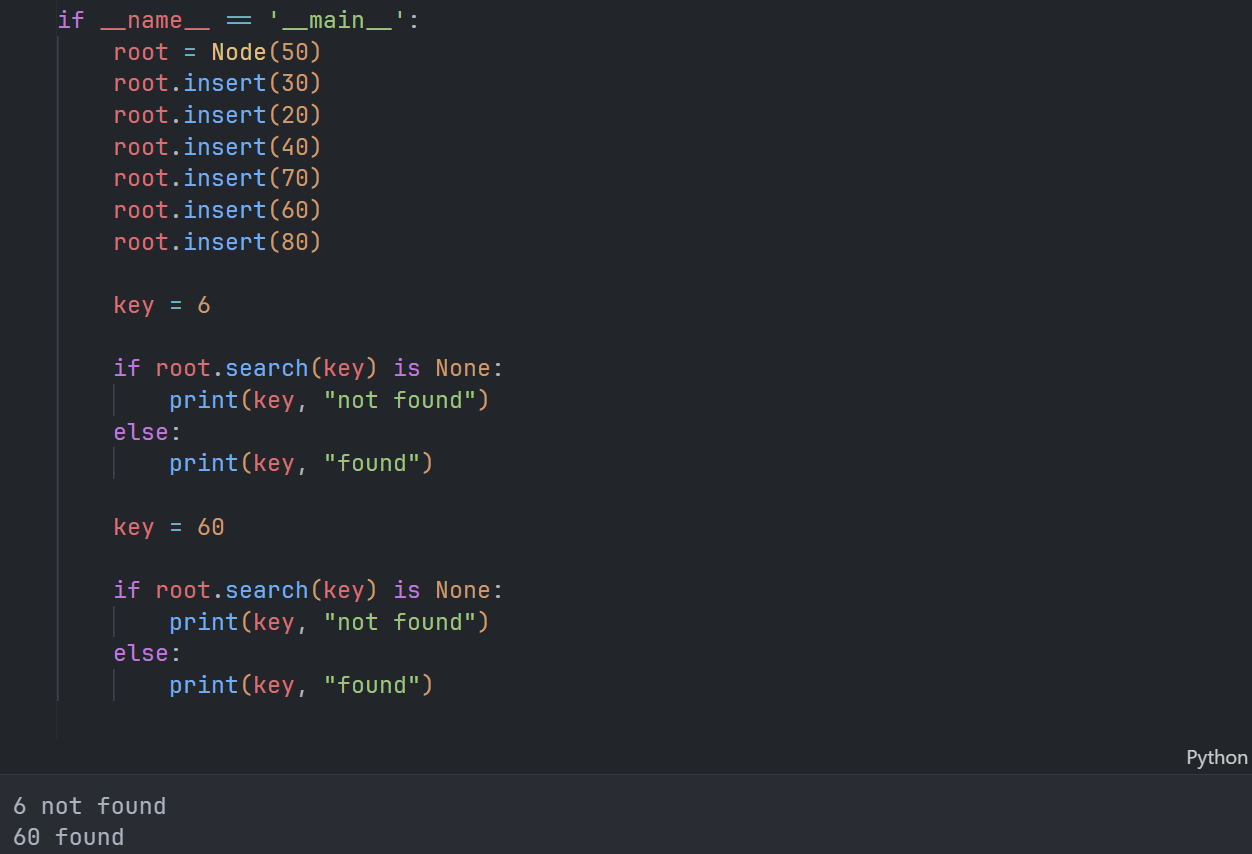
1. ‘class Node:’: Mendefinisikan kelas Node untuk merepresentasikan simpul dalam pohon biner.
2. ‘def \_\_init\_\_(self, data):’: Metode konstruktor untuk kelas Node. Ketika objek Node dibuat, metode ini akan dipanggil untuk menginisialisasi objek. Metode ini menerima parameter ‘data’, yang merupakan nilai yang akan disimpan dalam simpul.
3. ‘self.left = None’ dan ‘self.right = None’: Atribut ‘left’ dan ‘right’ digunakan untuk menyimpan referensi ke anak kiri dan anak kanan dari simpul saat ini. Awalnya, keduanya diatur menjadi ‘None’ karena simpul tersebut tidak memiliki anak saat dibuat.
4. ‘self.val = data’: Atribut ‘val’ digunakan untuk menyimpan nilai dari simpul itu sendiri. Nilai ini diambil dari parameter ‘data’ yang diterima oleh konstruktor.
5. ‘def insert(self, data):’: Metode ‘insert’ digunakan untuk menyisipkan elemen baru ke dalam pohon biner.
6. ‘def PrintTree(self):’: Metode ‘PrintTree’ digunakan untuk mencetak nilai-nilai dalam pohon biner secara inorder.
7. ‘def postorderTraversal(self, root):’: Metode ‘postorderTraversal’ digunakan untuk melakukan penelusuran post-order dari pohon biner. Ini akan mengembalikan daftar yang berisi nilai-nilai simpul dalam urutan post-order.
8. ‘res = []’: Variabel ‘res’ digunakan untuk menyimpan hasil penelusuran post-order.
9. ‘if root:’: Memeriksa apakah ‘root’ bukan ‘None’. Jika tidak, artinya masih ada simpul yang harus ditelusuri.
10. ‘res = self.postorderTraversal(root.left)’: Melakukan rekursi untuk menelusuri anak kiri dari ‘root’ dan menyimpan hasilnya dalam ‘res’.
11. ‘res = res + self.postorderTraversal(root.right)’: Melakukan rekursi untuk menelusuri anak kanan dari ‘root’ dan menambahkan hasilnya ke dalam ‘res’.
12. ‘res.append(root.val)’: Menambahkan nilai ‘root’ ke dalam ‘res’. Ini dilakukan setelah menelusuri kedua anak, karena pada penelusuran post-order, simpul saat ini diproses terakhir.
13. ‘return res’: Mengembalikan daftar yang berisi nilai-nilai dalam urutan post-order.
14. ‘root = Node(27)’: Membuat objek ‘root’ dengan nilai 27.
15. Menyisipkan beberapa nilai ke dalam pohon biner menggunakan metode ‘insert’.
16. ‘print(root.postorderTraversal(root))’: Mencetak hasil penelusuran post-order dari pohon biner dengan menggunakan metode ‘postorderTraversal’.
    * + 1. Hapus Elemen





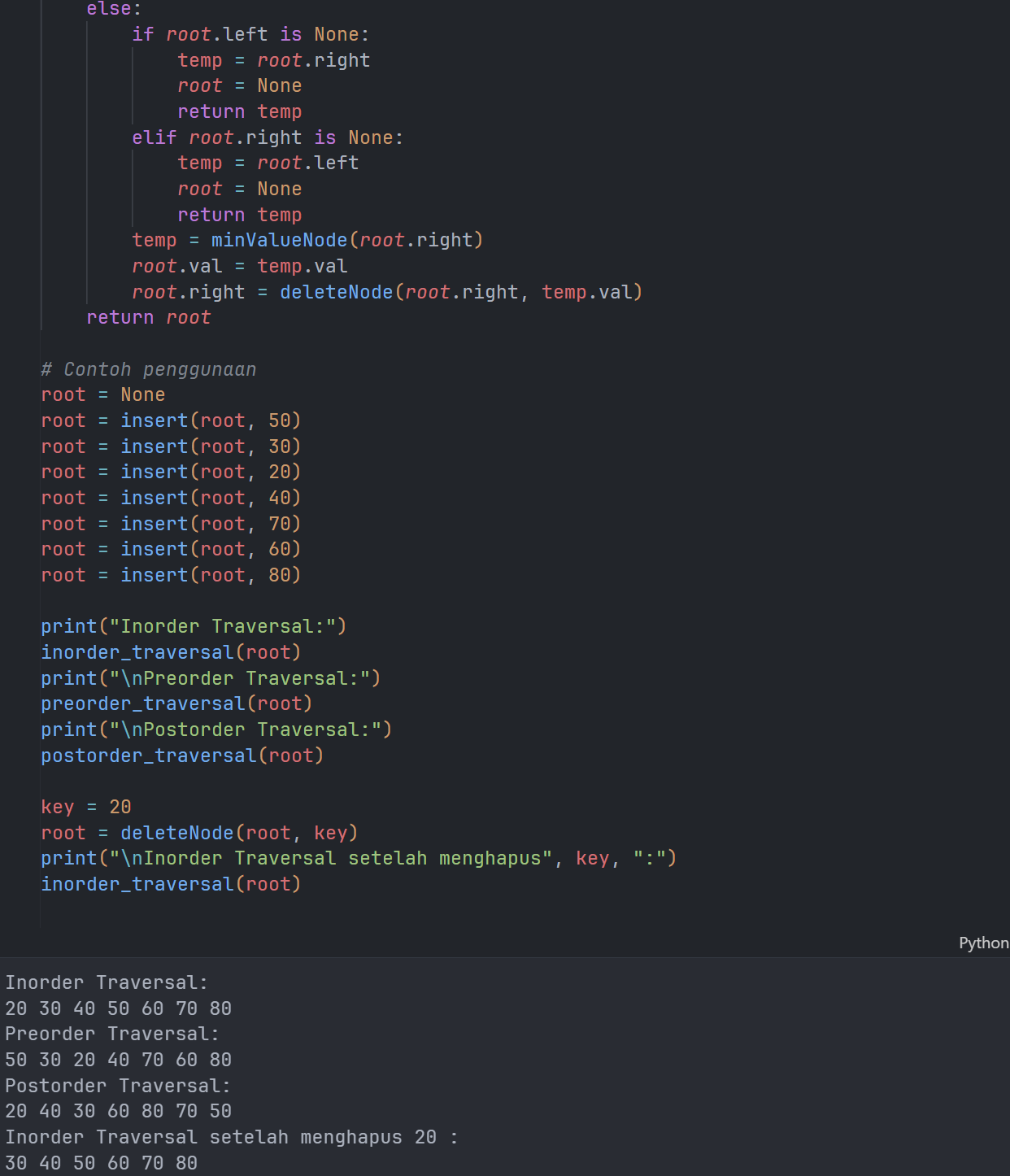
1. ‘class Node:’: Mendefinisikan kelas Node untuk merepresentasikan simpul dalam pohon biner.
2. ‘def \_\_init\_\_(self, data):’: Metode konstruktor untuk kelas Node. Ketika objek Node dibuat, metode ini akan dipanggil untuk menginisialisasi objek. Metode ini menerima parameter ‘data’, yang merupakan nilai yang akan disimpan dalam simpul.
3. ‘self.left = None’ dan ‘self.right = None’: Atribut ‘left’ dan ‘right’ digunakan untuk menyimpan referensi ke anak kiri dan anak kanan dari simpul saat ini. Awalnya, keduanya diatur menjadi ‘None’ karena simpul tersebut tidak memiliki anak saat dibuat.
4. ‘self.val = data’: Atribut ‘val’ digunakan untuk menyimpan nilai dari simpul itu sendiri. Nilai ini diambil dari parameter ‘data’ yang diterima oleh konstruktor.
5. ‘def inorder(self, temp):’: Metode ‘inorder’ digunakan untuk melakukan penelusuran inorder pada pohon biner dimulai dari simpul ‘temp’. Ini akan mencetak nilai-nilai simpul dalam urutan inorder.
6. ‘def deleteDeepest(self, root, d\_node):’: Metode ‘deleteDeepest’ digunakan untuk menghapus simpul terdalam (paling kanan) dari pohon biner. Ini akan membantu dalam operasi penghapusan simpul.
7. ‘def deletion(self, root, key):’: Metode ‘deletion’ digunakan untuk menghapus simpul dengan nilai ‘key’ dari pohon biner yang memiliki akar ‘root’. Ini akan memindai pohon biner menggunakan pendekatan BFS (Breadth-First Search) menggunakan antrian (queue).
8. ‘if root == None:’: Memeriksa apakah pohon kosong. Jika ya, langsung mengembalikan ‘None’.
9. ‘if root.left == None and root.right == None:’: Memeriksa apakah ‘root’ merupakan simpul daun. Jika ya, dan nilai ‘root’ sama dengan ‘key’, maka simpul tersebut dihapus dan mengembalikan ‘None’. Jika nilai ‘root’ tidak sama dengan ‘key’, maka simpul tersebut tetap dipertahankan.
10. ‘q = []’: Inisialisasi antrian.
11. ‘q.append(root)’: Memasukkan akar pohon ke dalam antrian.
12. ‘temp = None’: Inisialisasi variabel ‘temp’.
13. ‘while(len(q)):’: Melakukan loop selama antrian tidak kosong.
14. ‘temp = q.pop(0)’: Mengambil simpul dari depan antrian.
15. ‘if temp.val == key:’: Memeriksa apakah nilai simpul yang diambil sama dengan ‘key’.
16. ‘key\_node = temp’: Jika ya, menyimpan simpul tersebut ke dalam ‘key\_node’.
17. ‘if temp.left:’: Jika simpul memiliki anak kiri, masukkan anak kiri ke dalam antrian.
18. ’if temp.right:’: Jika simpul memiliki anak kanan, masukkan anak kanan ke dalam antrian.
19. ’if key\_node:’: Memeriksa apakah ‘key\_node’ tidak kosong (yaitu, simpul dengan nilai yang akan dihapus ditemukan).
20. ‘x = temp.val’: Menyimpan nilai dari simpul terakhir dalam ‘x’.
21. ’self.deleteDeepest(root, temp)’: Menghapus simpul terakhir dari pohon.
22. ’key\_node.val = x’: Menetapkan nilai ‘x’ ke simpul yang akan dihapus.
23. ’self.deleteDeepest(root, temp)’: Menghapus simpul yang dipindahkan ke simpul yang akan dihapus.
24. ’return root’: Mengembalikan akar pohon yang diperbarui setelah operasi penghapusan.
25. Bagian ‘if \_\_name\_\_ == '\_\_main\_\_':’ adalah bagian untuk menguji implementasi. Pohon biner dibuat, beberapa simpul ditambahkan, lalu sebuah simpul dihapus menggunakan metode ‘deletion’, dan pohon biner hasilnya dicetak dalam urutan inorder sebelum dan setelah penghapusan.
    * + 1. Pencarian Pada Binary Search Tree (BST)





1. ‘class Node:’: Mendefinisikan kelas Node untuk merepresentasikan simpul dalam pohon biner.
2. ‘def \_\_init\_\_(self, key):’: Metode konstruktor untuk kelas Node. Ketika objek Node dibuat, metode ini akan dipanggil untuk menginisialisasi objek. Metode ini menerima parameter ‘key’, yang merupakan nilai yang akan disimpan dalam simpul.
3. ‘self.left = None’ dan ‘self.right = None’: Atribut ‘left’ dan ‘right’ digunakan untuk menyimpan referensi ke anak kiri dan anak kanan dari simpul saat ini. Awalnya, keduanya diatur menjadi ‘None’ karena simpul tersebut tidak memiliki anak saat dibuat.
4. ‘self.val = key’: Atribut ‘val’ digunakan untuk menyimpan nilai dari simpul itu sendiri. Nilai ini diambil dari parameter ‘key’ yang diterima oleh konstruktor.
5. ‘def insert(self, key):’: Metode ‘insert’ digunakan untuk menyisipkan elemen baru ke dalam pohon biner.
6. ‘def search(self, key):’: Metode ‘search’ digunakan untuk mencari elemen dengan nilai tertentu dalam pohon biner.
7. ‘if self is None or self.val == key:’: Memeriksa apakah simpul saat ini ‘None’ atau memiliki nilai yang sama dengan ‘key’. Jika ya, mengembalikan simpul tersebut.
8. ‘if key < self.val:’ dan ‘if key > self.val:’: Membandingkan nilai ‘key’ dengan nilai ‘self.val’ untuk menentukan arah pencarian (apakah ke kiri atau ke kanan dari simpul saat ini).
9. ‘if \_\_name\_\_ == '\_\_main\_\_':’: Idiom Python untuk mengeksekusi beberapa kode hanya jika file dieksekusi secara langsung, bukan jika diimpor sebagai modul.
10. Pembuatan pohon biner dengan nilai-nilai tertentu menggunakan metode ‘insert’.
11. Pengujian pencarian elemen dengan nilai tertentu dalam pohon biner menggunakan metode ‘search’. Jika elemen ditemukan, mencetak "found", jika tidak, mencetak "not found".
12. **Latihan**

****

****

1. ‘class Node:’: Mendefinisikan kelas Node untuk merepresentasikan simpul dalam pohon biner.
2. ‘def \_\_init\_\_(self, key):’: Metode konstruktor untuk kelas Node. Menginisialisasi atribut left, right, dan val.
3. ‘def insert(root, key):’: Fungsi untuk menyisipkan elemen baru ke dalam pohon biner. Jika pohon kosong, maka fungsi akan membuat simpul baru sebagai akar. Jika tidak, fungsi akan mencari posisi yang sesuai untuk menyisipkan elemen baru.
4. ‘def inorder\_traversal(root):’, ‘def preorder\_traversal(root):’, ‘def postorder\_traversal(root):’: Fungsi-fungsi untuk melakukan penelusuran dalam urutan inorder, preorder, dan postorder, masing-masing dengan menggunakan rekursi.
5. ‘def minValueNode(node):’: Fungsi untuk menemukan simpul dengan nilai terkecil dalam pohon biner. Ini berguna saat menghapus simpul dengan dua anak.
6. ‘def deleteNode(root, key):’: Fungsi untuk menghapus simpul dengan nilai tertentu dari pohon biner. Terdapat beberapa kasus yang harus dipertimbangkan, seperti saat simpul yang akan dihapus memiliki satu anak atau dua anak.
7. Penggunaan contoh: Membuat pohon biner dengan beberapa nilai, mencetak penelusuran inorder, kemudian menghapus nilai tertentu dari pohon dan mencetak kembali penelusuran inorder untuk memverifikasi penghapusan.
8. **Kesimpulan**

Praktikum pemrograman Python di atas memberikan pemahaman yang kuat tentang implementasi dan operasi pada struktur data pohon biner. Melalui penggunaan kelas dan metode, kita dapat membuat, menyisipkan, mencari, dan menghapus simpul dalam pohon dengan efisien. Metode rekursi digunakan untuk melakukan traversal pada pohon, baik secara inorder, preorder, maupun postorder. Dengan demikian, kita dapat dengan mudah mengakses dan memanipulasi data dalam pohon sesuai kebutuhan. Pengujian fungsionalitas yang disertakan dalam praktikum memastikan bahwa operasi-operasi pada pohon berfungsi dengan benar dan memberikan hasil yang diharapkan.

Selain itu, praktikum ini juga memberikan penerapan konkret dari algoritma pencarian dalam konteks pohon biner. Dengan menggunakan teknik rekursi dan perbandingan nilai kunci, kita dapat dengan cepat menemukan elemen-elemen tertentu dalam pohon. Penerapan algoritma ini penting dalam berbagai aplikasi, seperti pencarian data, pengurutan, dan analisis struktur data. Dengan demikian, praktikum ini tidak hanya mengajarkan konsep dasar pemrograman Python, tetapi juga memberikan wawasan yang kuat tentang pemrosesan data dan algoritma yang digunakan dalam konteks struktur data yang kompleks seperti pohon biner.