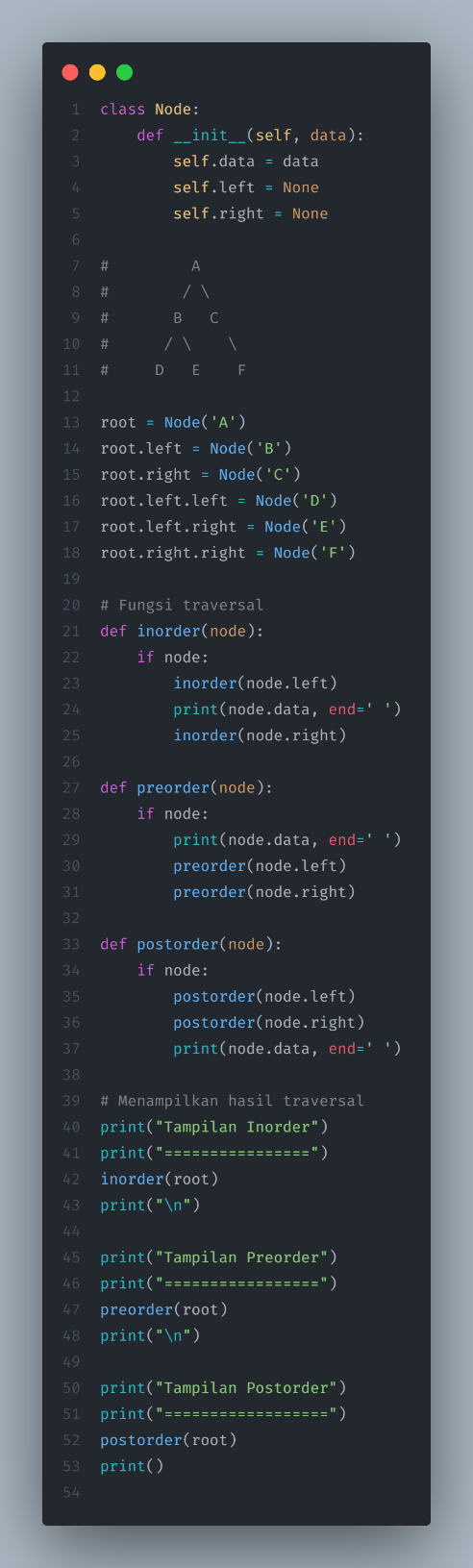
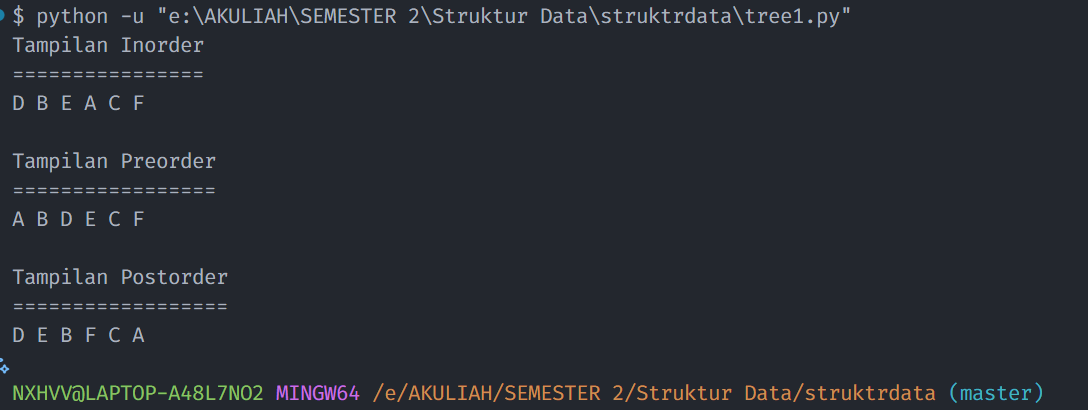
**LATIHAN 1 TREE**

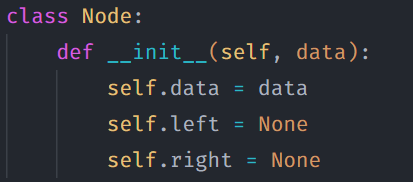
1. **Code**

****

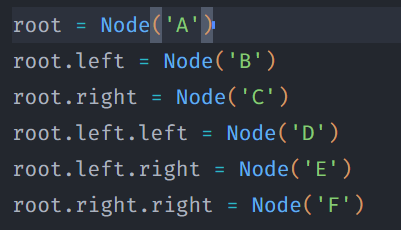
`x

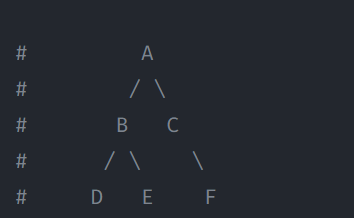
1. Hasil Running dari Latihan 1 Tree

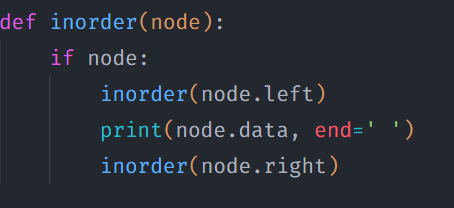


1. Penjelaskan Kode
2. **Kelas Node**

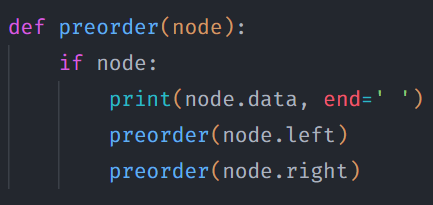
* Ini adalah definisi node pada binary tree.
* Setiap node memiliki
* Data: Nilai simpul
* Left : Anak kiri
* Right : Anak kanan

1. **Pembuatan Tree**

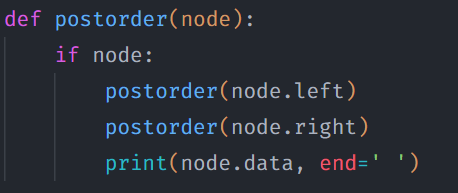
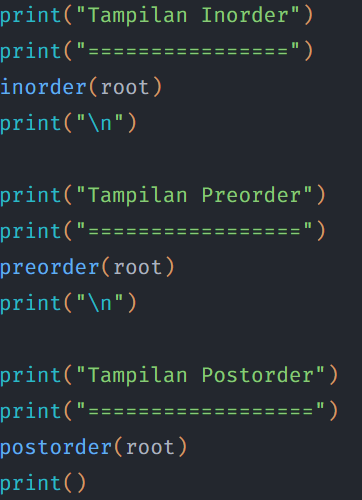
*  Struktur pohon

1. Inorder Traversal

* Urutan: **Kiri ➝ Root ➝ Kanan**  
  Output: D B E A C F

1. Preorder Traversal

* Urutan: **Root ➝ Kiri ➝ Kanan**  
   Output: A B D E C F

1. Postorder Traversal
   * Urutan: **Kiri ➝ Kanan ➝ Root**  
     Output: D E B F C A
2. Menampilkan Output
   * Menampilkan hasil penelusuran pohon untuk masing-masing metode.

**Latihan 2 Implementasikan BST dan operasi pencarian**

* Fungsi pencarian search(root, data)
* Masukkan data: [15, 10, 20, 8, 12, 17, 25]
* Cari nilai: 10 dan 99

1. **Codingan**

**A screen shot of a computer screen

Description automatically generated**

1. **Hasil Running dari Latihan 2 Implementasikan BST dan operasi pencarian**

**A black screen with white text

Description automatically generated**

1. **Penjelassan code**
2. **Class Node:**

**A screen shot of a computer code

Description automatically generated**

* class Node: Ini mendefinisikan sebuah kelas bernama Node. Dalam struktur data *tree*, setiap item data disimpan dalam sebuah *node* atau simpul.
* def \_\_init\_\_(self, data): Ini adalah metode *konstruktor* untuk kelas Node. Metode ini dipanggil setiap kali objek Node baru dibuat.
* self: Merujuk pada instance objek Node saat ini.
* data: Parameter ini akan menyimpan nilai aktual yang ingin Anda simpan di *node* ini (misalnya, angka, string, dll.).
* self.data = data: Baris ini menetapkan nilai data yang diterima sebagai argumen ke atribut data dari *node* ini.
* self.left = None: Ini menginisialisasi atribut left dari *node* ke None. Atribut left akan menjadi penunjuk (referensi) ke anak kiri dari *node* ini. Awalnya, tidak ada anak kiri.
* self.right = None: Ini menginisialisasi atribut right dari *node* ke None. Atribut right akan menjadi penunjuk (referensi) ke anak kanan dari *node* ini. Awalnya, tidak ada anak kanan.

1. **Class BST (Binary Search Tree)**

**A black background with white text

Description automatically generated**

* class BST: Ini mendefinisikan sebuah kelas bernama BST yang akan mengelola seluruh struktur *Binary Search Tree*.
* def \_\_init\_\_(self): Ini adalah metode *konstruktor* untuk kelas BST.
* self.root = None: Atribut root adalah penunjuk ke *node* paling atas dalam *tree*. Awalnya, ketika BST kosong, root diatur ke None.

1. **Metode insert**

**A computer screen shot of a code

Description automatically generated**

* **def insert(self, root, data):** Metode ini digunakan untuk menyisipkan nilai (data) baru ke dalam BST. Ini adalah fungsi rekursif.
* root: Ini adalah *node* saat ini yang sedang kita tinjau dalam proses penyisipan. (Catatan: ini adalah parameter, bukan self.root dari objek BST).
* data: Nilai yang akan disisipkan.
* **if root is None:** Ini adalah kasus dasar (base case) untuk rekursi. Jika *node* saat ini (root) adalah None, berarti kita telah menemukan lokasi yang tepat untuk menyisipkan nilai baru. Sebuah Node baru dibuat dengan data tersebut dan dikembalikan.
* **if data < root.data**: Jika nilai data yang akan disisipkan lebih kecil dari data yang ada di *node* saat ini (root.data), maka nilai tersebut harus disisipkan di *subtree* kiri.
* root.left = self.insert(root.left, data): Panggilan rekursif dilakukan pada anak kiri (root.left). Hasil dari panggilan rekursif (yaitu, *node* baru yang disisipkan atau *subtree* yang diperbarui) akan ditetapkan sebagai anak kiri dari *node* saat ini.
* **else:** Jika nilai data yang akan disisipkan lebih besar atau sama dengan data yang ada di *node* saat ini (root.data), maka nilai tersebut harus disisipkan di *subtree* kanan.
* root.right = self.insert(root.right, data): Panggilan rekursif dilakukan pada anak kanan (root.right). Hasilnya ditetapkan sebagai anak kanan dari *node* saat ini.
* **return root**: Setelah rekursi selesai dan *node* baru disisipkan, *node* root saat ini (yang mungkin sudah diperbarui dengan anak baru) dikembalikan ke panggilan sebelumnya.

1. **Metode search**

**A screen shot of a computer

Description automatically generated**

**def search(self, root, data):**: Metode ini mencari nilai (data) tertentu dalam BST. Ini juga fungsi rekursif.

* root: *Node* saat ini yang sedang kita tinjau.
* data: Nilai yang ingin dicari.
* **if root is None or root.data == data:**: Ini adalah kasus dasar.
  + Jika root adalah None, berarti kita telah mencapai akhir *tree* atau *subtree* tanpa menemukan nilai, jadi kembalikan None.
  + Jika root.data sama dengan data yang dicari, berarti kita telah menemukan *node* tersebut, jadi kembalikan root (node yang ditemukan).
* **if data < root.data:**: Jika nilai yang dicari lebih kecil dari data di *node* saat ini, maka cari di *subtree* kiri.
  + return self.search(root.left, data): Lakukan panggilan rekursif pada anak kiri.
* **return self.search(root.right, data)**: Jika nilai yang dicari lebih besar dari data di *node* saat ini, maka cari di *subtree* kanan. Lakukan panggilan rekursif pada anak kanan.

1. **Metode inorder**

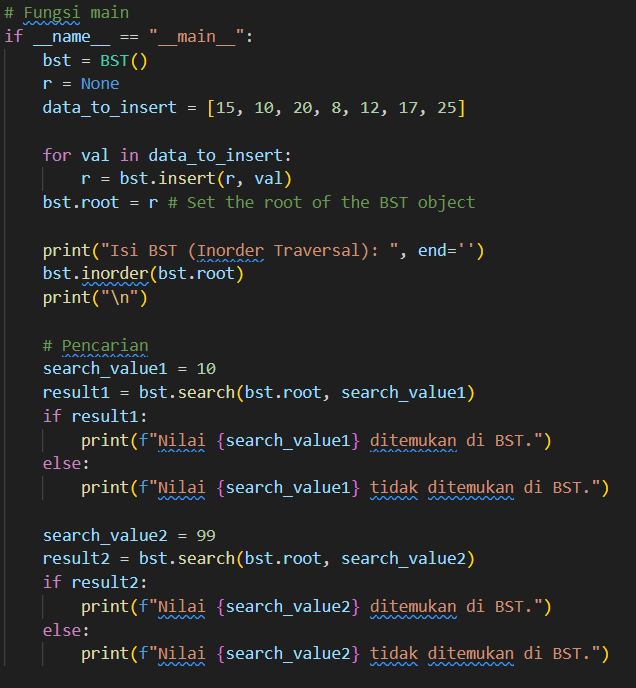
A screen shot of a computer code

Description automatically generated

**def inorder(self, root):** Metode ini melakukan *inorder traversal* pada BST. Traversal ini akan mengunjungi *node-node* dalam urutan terurut secara menaik (dari nilai terkecil ke terbesar). Ini adalah fungsi rekursif.

* root: *Node* saat ini yang sedang kita kunjungi.
* **if root:** Periksa apakah *node* saat ini tidak None. Jika None, tidak ada yang dilakukan.
  + self.inorder(root.left): Panggil inorder secara rekursif pada anak kiri. Ini memastikan semua *node* di *subtree* kiri dicetak terlebih dahulu.
  + print(root.data, end=' '): Setelah semua *node* di *subtree* kiri dicetak, cetak data dari *node* saat ini. end=' ' digunakan agar output berada dalam satu baris dengan spasi di antara angka.
  + self.inorder(root.right): Panggil inorder secara rekursif pada anak kanan. Ini memastikan semua *node* di *subtree* kanan dicetak setelah *node* saat ini.

1. **Bagian Utama (Fungsi Main)**



* **if \_\_name\_\_ == "\_\_main\_\_":** Blok kode ini akan dieksekusi hanya jika skrip dijalankan secara langsung (bukan diimpor sebagai modul).
* **bst = BST()**: Membuat sebuah instance baru dari kelas BST. Ini akan membuat objek BST kosong dengan root yang diatur ke None.
* **r = None**: Variabel r digunakan sebagai penunjuk sementara ke *root* *subtree* saat memanggil metode insert. Ini diperlukan karena metode insert bersifat rekursif dan mengembalikan *node* root yang diperbarui.
* **data\_to\_insert = [15, 10, 20, 8, 12, 17, 25]**: Ini adalah daftar nilai yang akan disisipkan ke dalam BST.
* **for val in data\_to\_insert:**: Loop ini mengiterasi setiap nilai dalam data\_to\_insert.
* **r = bst.insert(r, val)**: Di setiap iterasi, nilai saat ini (val) disisipkan ke dalam BST. Penting untuk diperhatikan bahwa r diperbarui dengan *root* yang dikembalikan dari insert. Ini memastikan bahwa r selalu mereferensikan *root* dari BST yang sedang dibangun.
* **bst.root = r**: Setelah semua nilai disisipkan, *root* akhir dari BST yang telah dibangun (yang disimpan dalam r) ditetapkan ke atribut bst.root dari objek BST. Ini penting agar operasi lain (seperti inorder dan search) dapat mengakses BST dengan benar.
* **print("Isi BST (Inorder Traversal): ", end='')**: Mencetak label untuk output *inorder traversal*.
* **bst.inorder(bst.root)**: Memanggil metode inorder untuk mencetak semua nilai dalam BST secara berurutan. bst.root digunakan sebagai *root* awal untuk traversal.
* **print("\n")**: Mencetak baris baru untuk pemisah.
* **Bagian Pencarian**:
* search\_value1 = 10 dan search\_value2 = 99: Mendefinisikan nilai yang akan dicari.
* result1 = bst.search(bst.root, search\_value1) dan result2 = bst.search(bst.root, search\_value2): Memanggil metode search untuk mencari nilai-nilai tersebut.
* **if result1: / if result2:**: Memeriksa apakah nilai yang dikembalikan oleh search adalah *node* (berarti ditemukan) atau None (berarti tidak ditemukan), kemudian mencetak pesan yang sesuai.