

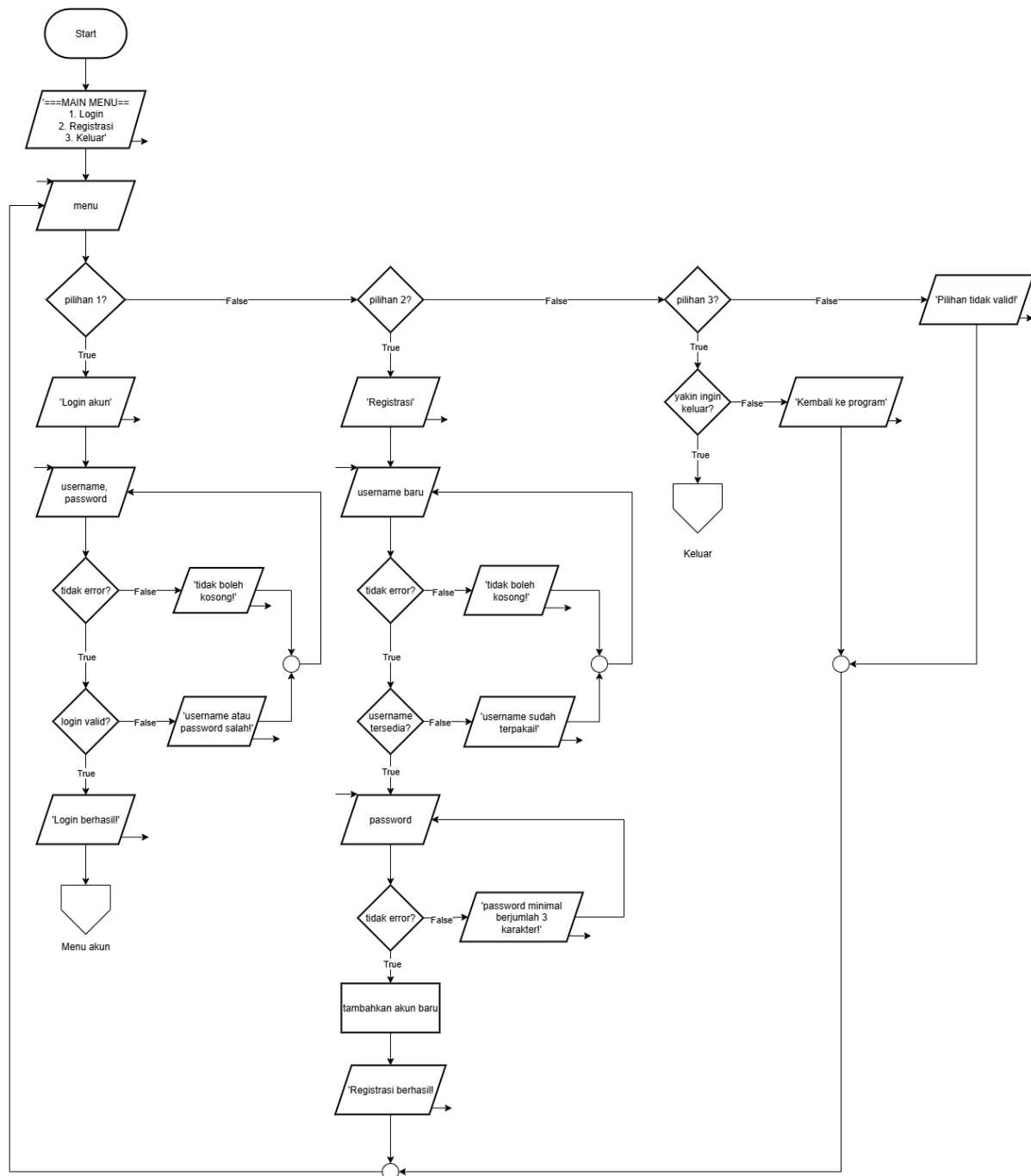
**LAPORAN PRAKTIKUM**  
**POSTTEST 5**  
**ALGORITMA PEMROGRAMAN DASAR**



**Disusun oleh:**  
**Muhamad Radja Nur Akbar (2509106012)**  
**Kelas (A1'25)**

**PROGRAM STUDI INFORMATIKA**  
**UNIVERSITAS MULAWARMAN**  
**SAMARINDA**  
**2025**

# 1. Flowchart

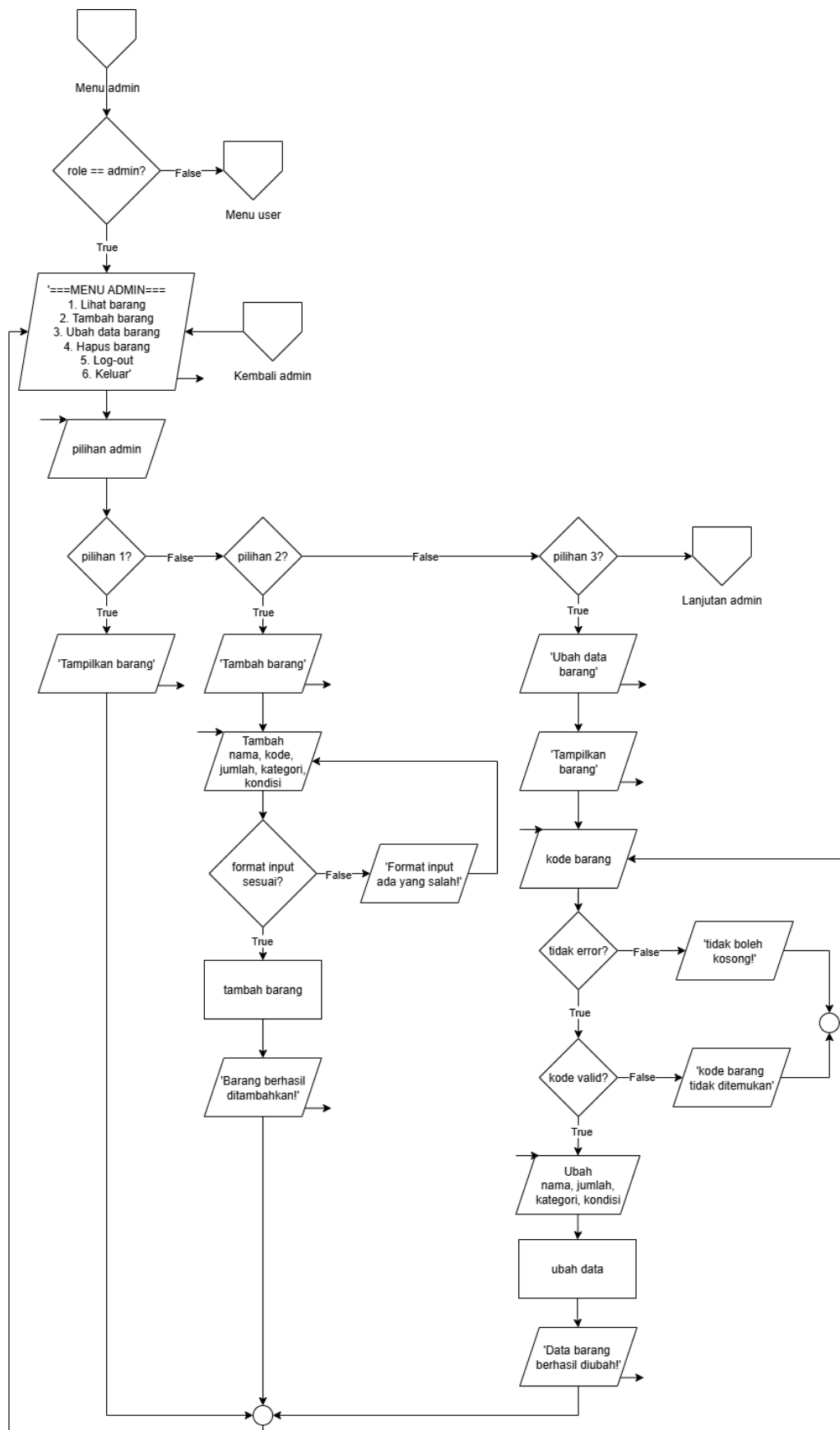


Gambar 1.1 Flowchart Main Menu

Flowchart dimulai dengan menampilkan pilihan dari *main menu*, jika pengguna memilih 1 atau *login* maka ia harus memasukkan *username* dan *password* dengan tepat. Jika kosong maka kembali *menginput* dan jika *username* dan *password* sesuai maka dilanjutkan dengan *menu admin*.

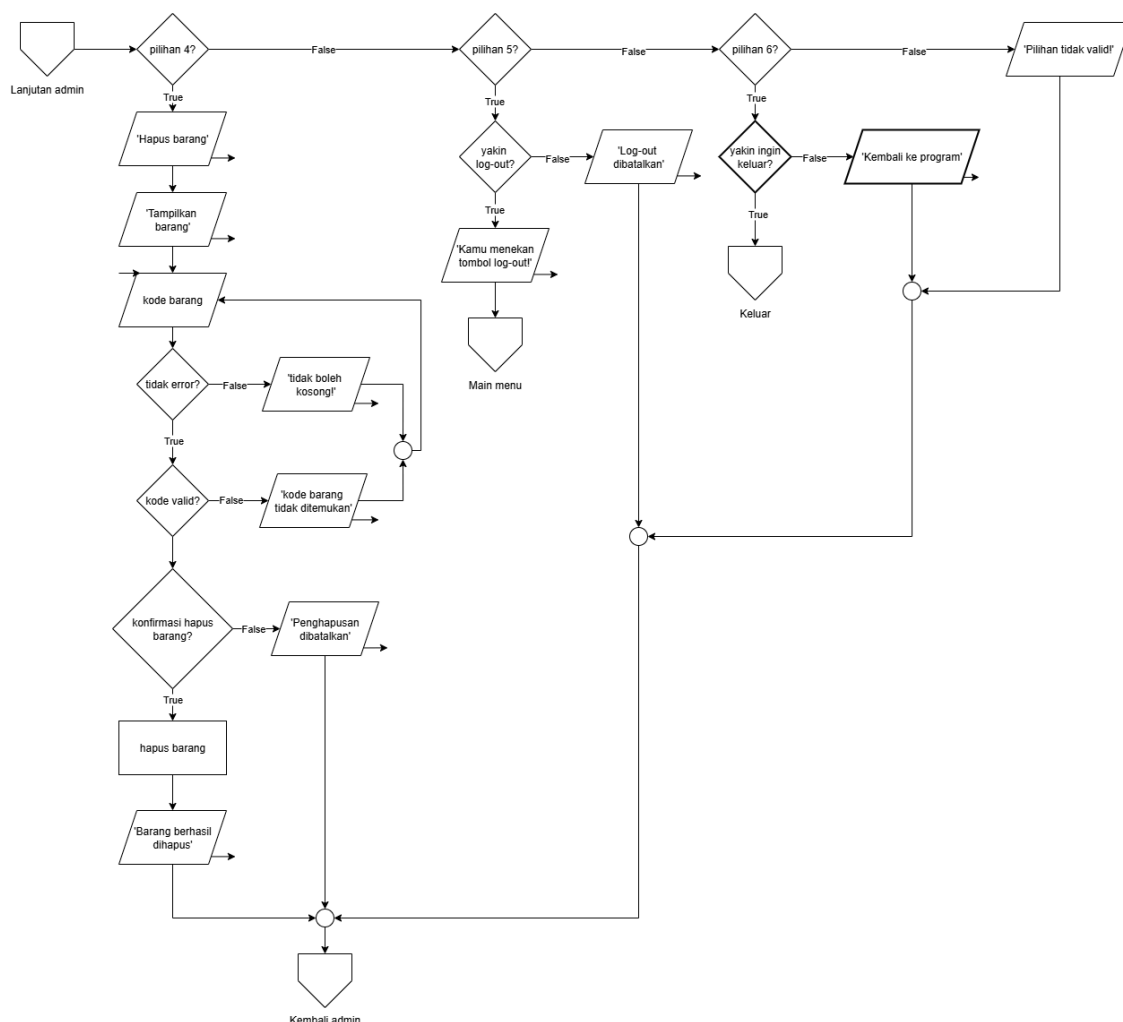
Kembali ke atas jika pengguna memilih 2 atau registrasi berarti ia membuat akun baru, setiap akun baru akan terdaftar sebagai *user*, atau *non-admin*. Proses dimulai dengan *menginput username* baru yang akan gagal jika *username* sudah digunakan. Lanjut dengan membuat *password* yang minimal berjumlah 3 karakter, seperti sebelumnya *input* tidak boleh kosong. Setelah *username* dan *password* baru memenuhi persyaratan maka akun baru berhasil ditambahkan dan kembali ke *main menu*.

Memilih 3 atau keluar maka program akan meminta konfirmasi sebelum betul-betul keluar dari program. *Input* yang tidak valid akan membawa kembali pengguna ke *main menu*.



Gambar 1.2 Flowchart Menu admin

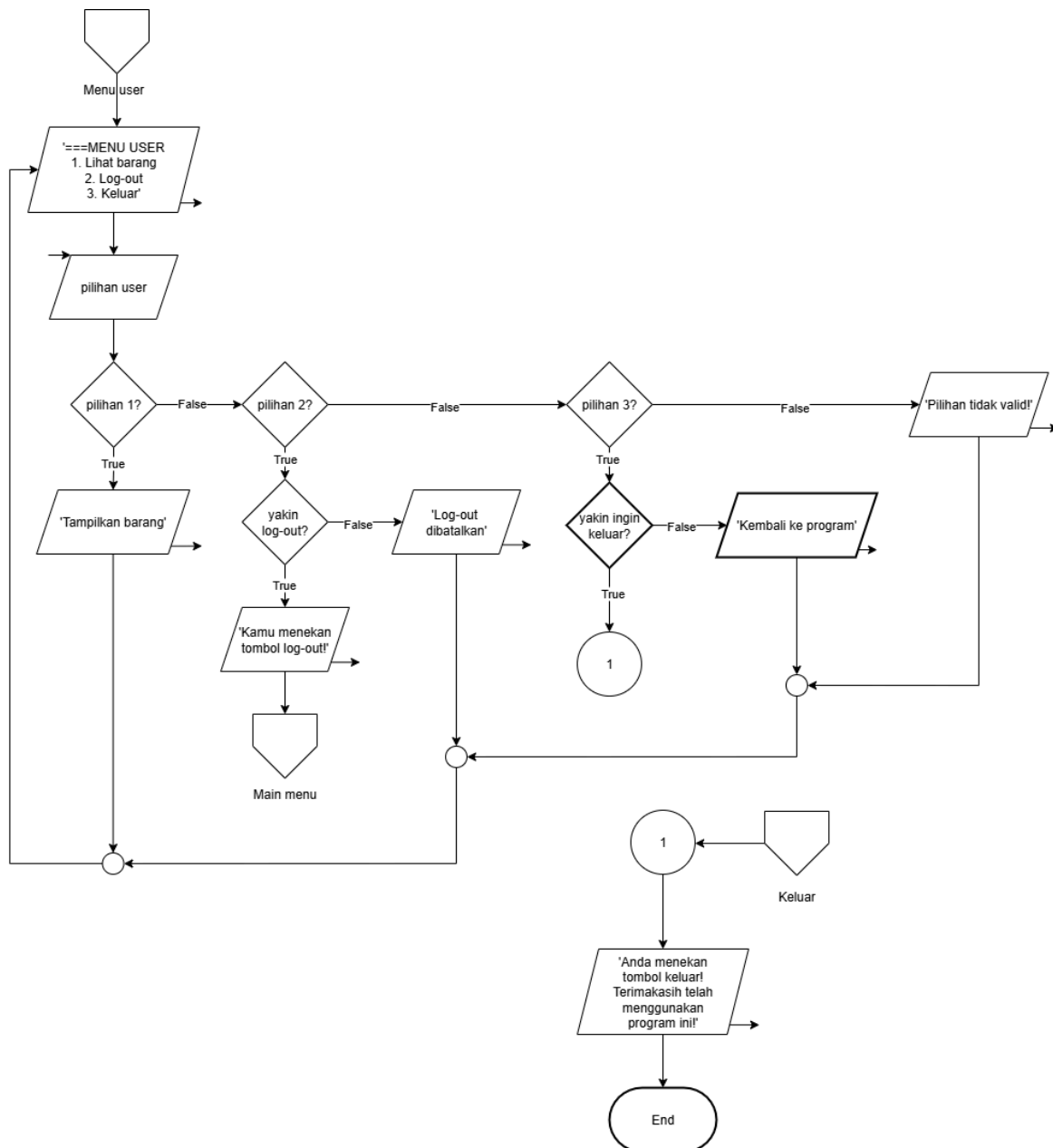
Pada *Gambar 1.2*, setelah berhasil login, *role* dari *username* akan dibaca. Seorang admin dapat melakukan 4 hal, melihat barang, menambah barang, mengubah data barang, atau menghapus barang. Jika memilih 1 atau lihat barang, daftar barang saat ini akan ditampilkan dan kembali ke *menu* admin. Memilih 2 atau menambah barang, pengguna akan diminta memasukkan data untuk barang baru seperti nama, kode, jumlah, kategori, dan kondisi. Jika format dari *inputan* sesuai maka barang akan ditambahkan dan kembali ke *menu* admin. Jika memilih 3 atau mengubah data barang, pengguna diminta memasukkan kode barang dari salah satu barang yang ada, jika ada maka berlanjut ke mengubah data barang yang diinginkan dan perubahan akan disimpan. Jika dilihat program akan kembali ke *input* kode barang jika kode yang dimasukkan tidak terdaftar di daftar barang.



*Gambar 1.3 Flowchart Lanjutan Admin*

Melanjutkan *flowchart* sebelumnya, pilihan ke-4 yaitu menghapus barang, seperti sebelumnya pengguna diminta untuk memasukkan kode barang yang benar dan meminta

konfirmasi sebelum betul-betul menghapus barang. Walaupun jawaban konfirmasi adalah tidak, pengguna akan dibawa kembali ke *menu admin* seperti pada *flowchart* sebelumnya. Pilihan yang lain ada *log-out* atau kembali ke *main menu* dan keluar dari program. Pilihan yang tidak valid akan kembali ke *menu admin*.



Gambar 1.4 Flowchart User dan End

Sebelumnya dijelaskan untuk setiap akun baru akan terdaftar sebagai *user*. *Login* sebagai akun baru dan *username* dan *password* benar akan membawa pengguna ke *menu user*. *User* hanya bisa melihat daftar barang jika memasukkan 1 pada pilihan *user*, 2 pilihan

lain adalah *log-out* dan keluar seperti pada *flowchart* sebelumnya. Seluruh *off-page reference* ‘Keluar’ pada *flowchart* saat ini dan sebelum-sebelumnya akan membawa pengguna ke *terminator end* dan program berakhir.

## 2. Deskripsi Singkat Program

Program ini memberi admin pilihan dalam mengakses barang-barang kantor sementara *user* hanya bisa melihat daftar barang. Program ini juga dilengkapi fitur registrasi untuk memberi pengguna akun baru untuk digunakan.

## 3. Source Code

Tugas CRUD ini berjudul ‘Pengelolaan Inventaris Barang Kantor’, sehingga barang-barang yang ada adalah barang-barang di sebuah kantor pada umumnya.

### A. Library Python

```
import os
import time
from prettytable import PrettyTable
```

Gambar 3.1.1 Library di Python

Tiga fitur ini merupakan *library* yang ada di Python. *Import OS* digunakan untuk operasi sistem pada terminal, seperti *clear*. Dengan mengetikkan kode di bawah:

```
os.system ('cls || clear')
```

Program otomatis menghapus terminal setelah eksekusi, kenapa ada ‘cls’ dan ‘clear’? Untuk ‘cls’ sendiri merupakan perintah membersihkan terminal untuk Windows, sedangkan di Linux dan MacOS menggunakan ‘clear’ untuk membersihkan terminal.

Lalu ada *import time*, merupakan *library* terkait dengan jeda waktu, dengan mengetik:

```
time.sleep(1)
```

Maka program akan diberi jeda 1 detik sebelum berlanjut ke kode berikutnya. Angka pada tanda kurung fleksibel, kalian bisa mengatur durasi jeda sesuka hati.

Terakhir *prettytable*, ini sudah bukan *library* bawaan Python melainkan harus diinstal terlebih dahulu dengan mengetikkan ‘pip install prettytable’ pada terminal. Ini berfungsi untuk menampilkan *output* dalam bentuk tabel tanpa harus membuat manual.

Tanpa menggunakan *import*, *library-library* tersebut tidak akan bisa dijalankan, pastikan kalian menambahkan *import* sebelum terjun ke program.

## B. List dan Nested List

```
inventaris = [  
    ['Monitor', 'BRG001', 7, 'Elektronik', 'Baik'],  
    ['Meja kayu', 'BRG002', 10, 'Perabot', 'Baik'],  
    ['Buku catatan', 'BRG003', 30, 'Alat tulis', 'Baik'],  
    ['Kursi plastik', 'BRG004', 20, 'Perabot', 'Baik'],  
    ['Printer', 'BRG005', 5, 'Elektronik', 'Baik'],  
    ['Pulpen', 'BRG006', 60, 'Alat tulis', 'Baik'],  
    ['PC', 'BRG007', 7, 'Elektronik', 'Baik']  
]  
users = [['radja', '012', 'admin']]
```

Gambar 3.2.1 List dan Nested List

List ‘inventaris’ di atas merupakan daftar barang awalan sebelum CRUD, setiap barang memiliki *indent* sendiri yang dimulai dari 0 sampai 6, karena menggunakan *nested list*, setiap elemen di dalamnya juga memiliki *indent* sendiri. Susunannya dimulai dari nama barang, kode barang, jumlah, kategori, dan kondisi. Tiga elemen pertama cukup penting dalam program berikutnya.

Selain *list* ‘inventaris’, ada juga *list* ‘users’, kenapa menggunakan *nested list* padahal hanya ada satu akun? Ini bertujuan untuk memungkinkan kita membuat akun baru. Susunannya dimulai *indent* 0 yaitu *username*, *password*, dan *role*. Seperti pada *flowchart*, hanya ada satu akun admin disini.



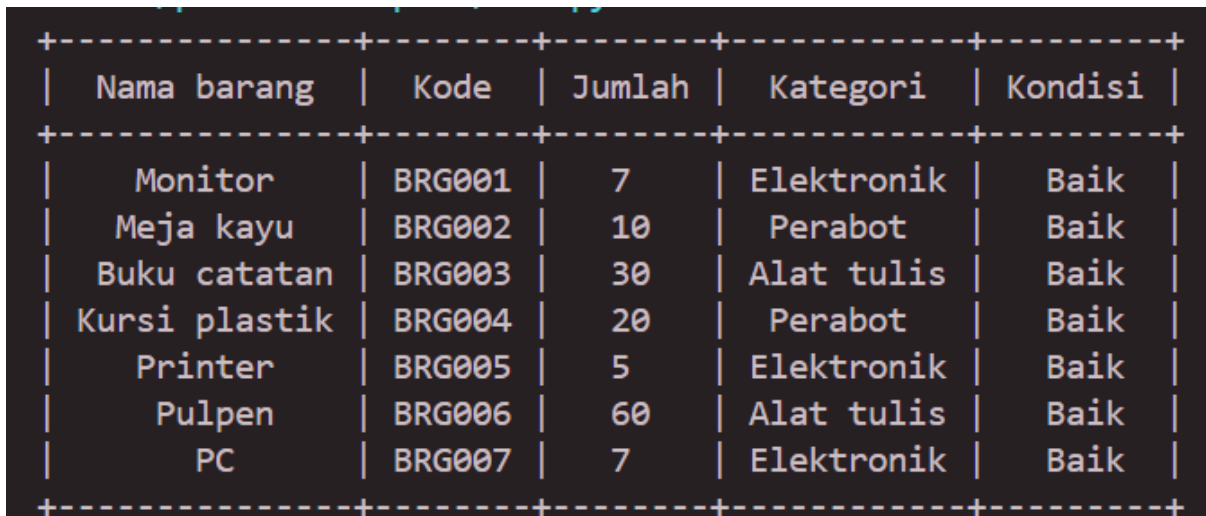
### C. PrettyTable

Dengan mengetikkan kode di bawah

```
tabel = PrettyTable ()
tabel.field_names = ['Nama barang', 'Kode', 'Jumlah', 'Kategori', 'Kondisi']
tabel.clear_rows()
for item in inventaris:
    tabel.add_row(item)
print(tabel)
```

Gambar 3.3.1 Kode Untuk Menampilkan PrettyTable

Pertama kita harus membuat variabel untuk memanggil *prettytable*, variabel ‘tabel’ akan berperan untuk itu. Lalu ada *field\_names* yang berfungsi menambah dan menentukan judul dari tiap kolom seperti yang dijelaskan sebelumnya, karena ada 5 elemen, maka akan terbuat 5 kolom. *clear\_rows* berfungsi menghapus semua baris dan menyisakan judul, ini bertujuan untuk mencegah tabel yang berulang ditampilkan di terminal. Dan ‘for item in inventaris’ untuk menampilkan setiap elemen dari *nested list*, tanpa ini elemen-elemen tersebut tidak akan ditampilkan. *add\_row* untuk menambahkan baris dan *print* untuk menampilkan tabel seperti pada gambar di bawah.



Nama barang	Kode	Jumlah	Kategori	Kondisi
Monitor	BRG001	7	Elektronik	Baik
Meja kayu	BRG002	10	Perabot	Baik
Buku catatan	BRG003	30	Alat tulis	Baik
Kursi plastik	BRG004	20	Perabot	Baik
Printer	BRG005	5	Elektronik	Baik
Pulpen	BRG006	60	Alat tulis	Baik
PC	BRG007	7	Elektronik	Baik

Gambar 3.3.2 Output PrettyTable Inventaris

### D. Fitur Login

```
while True:
    role = None
    loginn = False
    while True:
        print(''===Menu===
```

```
1. Login
2. Register
3. Keluar'''
    menu = input('Pilih menu:')
```

Gambar 3.4.1 While Loop Pembungkus dan Main menu

Sebelum *login*, program akan menampilkan opsi yang bisa dipilih, setiap opsi tentu memiliki program masing-masing yang dibungkus oleh *while True* tertinggi. Selain itu variabel dari 'role' dan 'loginn' akan ditetapkan sebagai *none* dan *False* untuk kode-kode berikutnya.

```
if menu == '1': #Menu login
    while True:
        print('Login akun')
        while True:
            usr = input('Username\t:')
            if usr.strip() != '':
                break
            else:
                print('Username tidak boleh kosong!')
```

Gambar 3.4.2 Input Username

Jika pengguna mengetikkan '1' pada 'menu', fitur *login* akan dijalankan, lalu kita ada *while True* untuk mengulang (*loop*) semua yang ada di fitur *main menu*. Kemudian ada *while True* yang membungkus fitur *login*, dan *while True* di dalam program meminta *input username*, disini fungsi *while True* terakhir tadi, jika dilihat ada 'if usr.strip() != ''' *strip* adalah salah satu *method* dalam Python yang berfungsi untuk menghapus semua spasi di awal dan akhir *string*. '!=' memiliki arti tidak sama dengan, dilanjutkan dengan '' atau *string* kosong. Kenapa menggunakan *strip*? Dan tidak langsung menggunakan '!='? Ini disebabkan jika tanpa *strip*, pengguna masih bisa membuat spasi yang dimana spasi juga dihitung sebagai karakter, ini membuat seakan-akan inputan itu kosong yang padahal ada spasi yang dihitung sebagai karakter, operasi tidak sama dengan masih membacanya sebagai *string* yang memiliki isi. Ini merupakan salah satu *error handling* untuk mencegah pengguna memasukkan nilai kosong terhadap suatu variabel *input*. Jika pengguna memasukkan sesuai aturan maka *while True* level 4 tadi akan berakhir dan kembali ke level sebelumnya. Jika tidak maka pengguna akan terus berada di level tersebut hingga memasukkan nilai yang sesuai. Harap diingat *error handling* ini akan digunakan terus hingga program berakhir.

```

while True:
    pw = input('Password\t:')
    if pw.strip() != '':
        break
    else:
        print('Password tidak boleh kosong!')

```

Gambar 3.4.3 Input Password

Seperti sebelumnya, *while True* ini untuk mencegah pengguna memasukkan *input* kosong saat diminta memasukkan *password*.

```

for u in users:
    if usr == u[0] and str(pw) == u[1]:
        loginn = True
        role = u[2]
        break

```

Gambar 3.4.4 Username dan Password Benar

Pada kode ini, fungsi ‘for u in users’ untuk membaca setiap elemen *nested list* yang ada di *list users*. Karena menggunakan *for*, *u[0]* dan *u[1]* akan dibaca sebagai *username* dan *password* yang ada di *list ‘users’*. Jika benar status ‘loginn’ yang awalnya *False* berubah menjadi *True* dan ‘role’ yang awalnya *none* berubah mengikuti *role* yang ada di *username* yang dimasukkan. Anggap saja kita memasukkan ‘radja’ sebagai *usr* dan ‘012’ sebagai *pw*, karena sesuai maka nilai dari kedua variabel tadi akan berubah, ‘role’ mengikuti *role* dari ‘radja’. Setelah itu *break*.

```

if loginn:
    os.system('cls || clear')
    print('Login berhasil!')
    time.sleep(1)
    break
else:
    os.system('cls || clear')
    print('Username atau Password salah!')
if loginn:
    break

```

Gambar 3.4.5 Dua Break

Program otomatis menghitung nilai 'loginn' sebagai *True* tanpa perlu mengetikkan operasi logika '==', jika *usr* dan *pw* salah maka kembali meminta pengguna untuk *menginput* yang sesuai. Kenapa ada dua 'if loginn' yang mengandung *break*? Sebelumnya terdapat dua *while True*, level 2 yang *nge-loop main menu* dan level 3 yang *nge-loop fitur login*. *Break* pertama untuk level 3 dan *break* kedua untuk level 2. Kedua *break* ini diperlukan untuk lanjut ke *menu admin* nantinya.

### E. Fitur Registrasi (CREATE)

```
elif menu == '2': #Menu register
    os.system('cls || clear')
    print('Registrasi')
    while True:
        cek_ada_usr = False
        while True:
            usr_baru = input('Username baru\t:')
            if usr_baru.strip() != '':
                break
            else:
                print('Username tidak boleh kosong!')
```

Gambar 3.5.1 Fitur Registrasi

Jika pengguna *menginput* '2', maka akan lanjut ke fitur registrasi. Fitur ini memungkinkan kita membuat akun baru yang bisa *diloginkan* nantinya. Dengan sedikit catatan setiap akun baru akan dihitung sebagai *user* dan bukan *admin*. Seperti sebelumnya menggunakan *error handling* untuk mencegah *input* kosong. Ada tambahan variabel 'cek\_ada\_usr' dengan nilai awalan *False*, bisa dilihat pada kode berikutnya.

```
for u in users:
    if usr_baru == u[0]:
        cek_ada_usr = True
        break
if cek_ada_usr:
    print('Username sudah terpakai')
else:
    while True:
        pw_baru = input('Password baru\t:')
        if len(pw_baru) < 3:
            print('Password minimal berjumlah 3 karakter!')
        else:
            break
```

```

users.append([usr_baru, pw_baru, 'user'])
os.system('cls || clear')
print('Register berhasil!')
break

```

Gambar 3.5.2 Lanjutan Registrasi

Menggunakan *for* untuk membaca elemen pada *nested list*. Saat *username* baru ditambahkan dan ternyata ada *username* yang sama, maka pengguna diminta memasukkan *username* yang berbeda. Karena ada yang sama nilai dari 'cek\_ada\_usr' menjadi *True* dan kode 'if cek\_ada\_usr:' dijalankan. Jika *username* aman, maka lanjut memasukkan *password* baru. *Password* memiliki aturan, yaitu jumlah karakter atau elemen dalam *password* minimal 3, karena sudah ada kode tersebut kita tidak perlu lagi menggunakan '!=' karena otomatis *password* kosong akan dihitung sebagai elemen kurang dari 3, kode ini menggunakan *length* atau *len* untuk menghitung jumlah elemen yang ada di dalam suatu objek (*str*, *list*, dll). Setelah memenuhi semua syarat, *username* dan *password* yang baru akan ditambah ke dalam *list* 'users' dengan menggunakan *append*, yaitu *method* pada *list* untuk menambah elemen atau *nested list* baru ke belakang sebuah *list*. Karena ada [ ] di dalam *users.append*, maka akun baru dihitung sebagai *nested list* baru. Setelah itu ada *break* untuk mengakhiri *loop* registrasi.

```

elif menu == '3': #Keluar
    os.system('cls || clear')
    konfirmasiKeluar = input('Yakin ingin keluar?(y/n):')
    if konfirmasiKeluar == 'y':
        os.system('cls || clear')
        print('Anda menekan tombol keluar! Terima kasih telah
menggunakan program ini!')
        exit() #Keluar dari program
    elif konfirmasiKeluar == 'n':
        print('Kembali ke program')
    else:
        print('Pilihan tidak valid!')

else: #Pilihan di luar opsi
    os.system('cls || clear')
    print('Pilihan tidak valid!')

```

Gambar 3.5.3 Pilihan 3 dan Pilihan Tidak Valid

Pilihan selanjutnya adalah keluar dari program, sebelum keluar pengguna akan diminta konfirmasi sebelum akhirnya keluar dari program sepenuhnya. Disini kita menggunakan `exit ()` yang berfungsi untuk keluar langsung dari program.

## F. Menu Admin

```
while True:
    if role == 'admin': #Menu admin
        print(f'Halo, {usr}!')
        print(''===MENU ADMIN===')
        1. Lihat barang
        2. Tambah barang
        3. Ubah data barang
        4. Hapus barang
        5. Log-out
        6. Keluar''')
        menu_admin = input('Pilih menu:')
```

Gambar 3.6.1 Pilihan Menu Admin

Sebelumnya saat berhasil *login*, ada dua *break* untuk mengakhiri *loop main menu* dan *loop login*. Karena kedua *loop* telah berhenti, program berlanjut ke *menu* sesuai dengan *role* dari akun yang *diloginkan*. Kita mulai dari admin yang memiliki empat wewenang yang menjadi landasan dari CRUD. Pengguna diminta untuk *input* pilihan yang diinginkan.

## G. Melihat Daftar Barang (READ)

```
if menu_admin == '1': #Lihat barang, READ
    os.system ('cls || clear')
    print('Daftar barang')
    tabel.clear_rows() #Membersihkan tabel yang berulang

    for item in inventaris:
        tabel.add_row(item)
    print(tabel)
    banyak_barang = len(inventaris)
    print(f'Jumlah barang: {banyak_barang}')
```

Gambar 3.7.1 Melihat Daftar Barang

Pilihan pertama cukup sederhana karena hanya menampilkan daftar barang saat ini, dengan menggunakan *External Library PrettyTable* seperti yang dijelaskan di sub-bab C, kita bisa melihat apa saja barang-barang yang ada di tabel, termasuk setelah kita menambah barang, mengubah data barang, dan menghapus barangnya nantinya. Ini juga dilengkapi

dengan *length* untuk menghitung berapa banyak barang yang ada di *list* 'inventaris' dengan memanfaatkan variabel 'banyak\_barang'.

## H. Menambah Barang Baru (CREATE)

```
elif menu_admin == '2':
    os.system('cls || clear')
    print('Tambah barang')

    while True: #Nama barang baru
        barang_baru = input('Nama barang\t:')
        if barang_baru.strip() != '':
            break
        else:
            print('Nama barang tidak boleh kosong!')
```

Gambar 3.8.1 Menambah Nama Barang Baru

Setiap barang memiliki lima atribut atau *field*. Setiap program untuk menambah *field* selalu diawali dengan *while True* untuk mencegah *input* kosong dan menggunakan variabel sendiri untuk menampung nilai yang dimasukkan. Pertama ada nama barang baru seperti pada kode di atas.

```
while True: #Kode barang baru
    cek_kode_barang = False
    print('Contoh kode barang: BRG001, BRG002')
    while True:
        kode_barang = input('Kode barang\t:')
        if kode_barang.strip() != '':
            break
        else:
            print('Kode barang tidak boleh kosong')
    for j in inventaris:
        if kode_barang == j[1]:
            cek_kode_barang = True
            break
    if cek_kode_barang:
        print('Kode barang sudah digunakan! Gunakan kode lain')
    else:
        break
```

Gambar 3.8.2 Menambah Kode Barang Baru

Berikutnya pengguna diminta untuk memasukkan kode barang baru. Sebelum itu kita menggunakan tipe data *boolean* lagi untuk mencegah kode barang baru yang dimasukkan

sama dengan kode yang sudah ada. Jika ada yang sama variabel 'cek\_kode\_barang' menjadi *True* yang akan menjalankan kode 'if cek\_kode\_barang:'

```
while True:
    jumlah_input = input('Jumlah barang\t:')
    if jumlah_input.isdigit() and int(jumlah_input) > 0:
        jumlah_barang = int(jumlah_input)
        break
    else:
        print('Jumlah harus tidak boleh kosong dan harus berupa angka lebih dari 0!')
```

Gambar 3.8.3 Jumlah Barang Baru

*Field* berikutnya adalah jumlah, tidak seperti sebelumnya, karena jumlah melibatkan *integer*, jumlah minimal yang harus dimasukkan saat menambah barang adalah satu alias lebih dari nol. Disini walaupun variabel input 'jumlah\_input' bukan *integer*, nantinya akan dirubah menjadi tipe data tersebut setelah melewati aturan sebelumnya. *isdigit* memastikan semua nilai *input* berupa angka. Karena menggunakan *and* jika salah satu salah *print* yang berada di *else* akan ditampilkan dan kembali ke *input* jumlah barang.

```
while True:
    kategori_barang = input('Kategori barang\t:') #Kategori
    barang baru
    if kategori_barang.strip() != '':
        break
    else:
        print('Kategori barang tidak boleh kosong')
    while True:
        kondisi_barang = input('Kondisi barang\t:') #Kondisi
        barang baru
        if kondisi_barang.strip() != '':
            break
        else:
            print('Kondisi barang tidak boleh kosong!')
```

Gambar 3.8.4 Kategori dan Kondisi Barang Baru

Proses terus berlanjut hingga kategori dan kondisi barang. Dengan kode yang sama seperti sebelumnya, langkah terakhir yaitu menambah barang tersebut ke dalam *list* 'inventaris'.

```
inventaris.append([barang_baru, kode_barang, jumlah_barang,
    kategori_barang, kondisi_barang])
```



```
os.system('cls || clear')
print('Barang berhasil ditambahkan!')
```

Gambar 3.8.5 Menambah Barang ke Dalam List

Setelah semua nilai *field* terisi, barang baru akan ditambah ke belakang *list* 'inventaris' menggunakan *inventaris.append*. Jangan lupa menyetikkan susunannya dengan benar. Dengan ini maka barang yang baru telah ditambahkan ke *list*, kalian bisa membuktikannya dengan mengetik '1' setelah kembali ke *menu* admin untuk melihat tabelnya.

## I. Mengubah Data Barang (UPDATE)

Di bagian ini, admin memiliki wewenang untuk mengubah data barang yang sudah ada, kecuali kode. Karena kode ini akan digunakan untuk memanggil barang yang ingin diubah datanya.

```
elif menu_admin == '3':
    os.system('cls || clear')
    print('Ubah data barang')
    print('Daftar barang')
    tabel.clear_rows()

    for item in inventaris:
        tabel.add_row(item)
    print(tabel)
    while True:
        #Membaca apakah kode barang yang diinput ada di list
        cari_kode = input('Masukkan kode barang yang ingin
        diubah: ')

        if cari_kode.strip() != '':
            break
        else:
            print('Tidak boleh kosong!')
    os.system('cls || clear')
    ditemukan = False

    for ubah_barang in inventaris:
        if ubah_barang[1] == cari_kode:
            ditemukan = True
            print(f'''
            Data ditemukan:
            Nama\t\t: {ubah_barang[0]}
            Kode\t\t: {ubah_barang[1]}''')
```

```
Jumlah\t\t: {ubah_barang[2]}
Kategori\t: {ubah_barang[3]}
Kondisi\t\t: {ubah_barang[4]}''')
```

Gambar 3.9.1 Memanggil Barang dengan Kode Barang

Pertama tabel ‘inventaris’ akan ditampilkan untuk mempermudah pengguna mencari kode barang yang ingin diubah datanya.

Setelah itu *menginput* kode barang dengan *error handling input* kosong, kode barang akan dibandingkan dengan setiap kode barang yang ada di ‘inventaris’

Jika ditemukan ada kode barang yang sesuai dengan ‘cari\_kode’ yang ada di variabel ‘kode\_barang’, maka nilai ‘ditemukan’ menjadi True, tujuannya akan ditunjukkan di akhir Ubah Data Barang. Kemudian barang akan ditampilkan beserta dengan semua *fieldnya*.

**Catatan**, variabel ‘ubah\_barang’ akan terus digunakan hingga program ubah data barang berakhir, karena menggunakan *for*, variabel ini menyimpan setiap satu baris pada ‘inventaris’. Begitu ada kecocokan, baris tersebut akan dipakai hingga proses pengubahan data/*field* selesai.

```
print('''Pilih data yang ingin diubah:
1. Ubah nama barang
2. Ubah jumlah barang
3. Ubah kategori
4. Ubah kondisi''')

pilihan_ubah = input('Pilih apa yang ingin diubah: ')

if pilihan_ubah == '1':
    while True:
        ubah_barang[0] = input('Ubah nama barang: ')
        if ubah_barang[0].strip() != '':
            break
        else:
            print('Nama barang tidak boleh kosong!')
    os.system('cls || clear')
    print('Nama barang berhasil diubah!')
```

Gambar 3.9.2 Ubah Nama Barang

Setelah kode barang ditemukan, langkah berikutnya memilih data/*field* apa yang ingin diubah. Jika memilih ‘1’, maka nama barang bisa diubah, karena dalam ‘inventaris’ nama barang berada pada *indent nol*, maka ditulis ‘ubah\_barang[0]’

```
elif pilihan_ubah == '2':
    os.system('cls || clear')
```

```

        print('Ubah jumlah barang: ')
        print(f'Jumlah barang saat ini:
{ubah_barang[2]}')

        print('''Pilih:
1. Tambah jumlah barang
2. Kurangi jumlah barang''')

```

Gambar 3.9.3 Ubah Jumlah Barang

Berikutnya mengubah *field* jumlah barang. Disini ada opsi menambahkan atau mengurangi. Jumlah barang yang dipilih akan ditampilkan sebagai syarat di kode selanjutnya.

```

        while True:
            tamkur = input('Pilih (1/2): ') #tamkur =
tambah kurang

            if tamkur in ['1', '2']:
                break
            else:
                print('Pilihan tidak valid!')
        while True:
            jumlah_tamkur = input('Masukkan jumlah: ')
            if jumlah_tamkur.isdigit() and
int(jumlah_tamkur) > 0:

                proses_operasional = int(jumlah_tamkur)

                if tamkur == '1':
                    ubah_barang[2] += proses_operasional
                    print('Jumlah barang berhasil
ditambahkan!')

                    break

                elif tamkur == '2':
                    if ubah_barang[2] >=
proses_operasional:
                        ubah_barang[2] -=
proses_operasional

                        print('Jumlah barang berhasil
dikurangi!')

                        break
                    else:
                        print('Jumlah terlalu besar
untuk dikurangi')

                else:
                    print('Harus berupa angka lebih dari
0!')

```

Gambar 3..9.4 Proses Menambang dan Mengurangi Jumlah Barang

Saya menggunakan variabel 'tamkur' yang merupakan singkatan dari tambah kurang, variabel ini menyimpan nilai *input* dari pilihan sebelumnya. Disini percabangan nilai 'tamkur' menggunakan proses yang sedikit berbeda, menggunakan *list* yang berisi '1' dan '2', jika jawaban 'tamkur' adalah salah satu dari dua elemen tersebut, maka *loop* nilai 'tamkur' akan berhenti dan menyimpan elemen yang *diinput*, jika *input* di luar elemen *list*, maka akan ada pesan 'pilihan tidak valid!'

Sebelum menuju proses operasional, pengguna diminta untuk *menginput* jumlah barang yang akan dioperasikan. Jika 'jumlah\_tamkur' berupa angka dan lebih dari nol, akan ada variabel 'proses\_operasional' yang menyimpan jumlah dari 'jumlah\_tamkur' sebagai *integer*, karena sebelumnya jumlah berupa *string*.

Masuk ke proses operasional penjumlahan, jumlah barang saat ini akan ditambah dengan 'proses\_operasional' dan mengakhiri *loop* ubah data barang. Begitu juga dengan pengurangan, perbedaan ada di jumlah awal barang jika kurang dari 'jumlah\_tamkur', maka proses akan kembali ke *input* 'jumlah\_tamkur', ini untuk mencegah jumlah menjadi negatif. *Else* penutup untuk ubah jumlah barang berpasangan dengan *if* yang nilai 'jumlah\_tamkur' harus berupa angka dan lebih dari nol, jika ini terjadi maka pengguna harus *menginput* kembali nilai yang tepat.

```
elif pilihan_ubah == '3': #Update kategori barang
    os.system('cls || clear')
    while True:
        ubah_barang[3] = input('Ubah kategori
barang: ')

        if ubah_barang[3].strip() != '':
            break
        else:
            print('Kategori barang tidak boleh
kosong!')

        print('Kategori berhasil diubah!')

    elif pilihan_ubah == '4': #Update kondisi barang
        os.system('cls || clear')
        while True:
            ubah_barang[4] = input('Ubah kondisi barang:
')

            if ubah_barang[4].strip() != '':
                break
            else:
                print('Kondisi barang tidak boleh
kosong!')

            print('Kondisi barang berhasil diubah')
```

```

        else:
            os.system('cls || clear')
            print('Pilihan tidak valid')
            break
    if not ditemukan:
        os.system('cls || clear')
        print('Kode barang tidak ditemukan')

```

Gambar 3.9.5 Lanjut Proses Ubah Data Barang

Pilihan ke-3 dan ke-4 sama saja dengan pilihan ke-1. Pilihan ini mengubah kategori atau kondisi barang yang dipilih. Jika pilihan di luar keempat *field*, maka akan kembali ke *menu* admin. Masih ingat dengan variabel ‘ditemukan’? Jika kode barang tidak dalam ‘inventaris’, maka akan kembali ke *menu* admin.

## J. Menghapus Barang (DELETE)

```

elif menu_admin == '4': #DELETE, menghapus barang
    os.system('cls || clear')
    print('Daftar barang')

    tabel.clear_rows()

    for item in inventaris:
        tabel.add_row(item)
    print(tabel)
    print('Hapus barang')

    while True:
        hapus = input('Masukkan kode barang yang ingin dihapus:
')

        if hapus.strip() != '':
            break
        else:
            print('Tidak boleh kosong!')
    ketemu = False

    for hapus_barang in inventaris:
        if hapus_barang[1] == hapus:
            ketemu = True
            print(f'''
Data ditemukan:
Nama\t\t: {hapus_barang[0]}
Kode\t\t: {hapus_barang[1]}

```

```

Jumlah\t\t: {hapus_barang[2]}
Kategori\t: {hapus_barang[3]}
Kondisi\t\t: {hapus_barang[4]}'')
        konfirmasi = input('Konfirmasi hapus barang?(y/n):')
        if konfirmasi == 'y':
            os.system('cls || clear')
            inventaris.remove(hapus_barang)
            print('Barang berhasil dihapus')
        elif konfirmasi == 'n':
            os.system('cls || clear')
            print('Penghapusan dibatalkan')
        else:
            os.system('cls || clear')
            print('Tidak dalam pilihan!')
            break
    if not ketemu:
        print('Kode barang tidak ditemukan!')

```

Gambar 3.10.1 Proses Menghapus Barang

Proses menghapus barang mirip dengan mengubah data barang, pertama kita *menginput* kode barang yang kemudian kode yang sesuai disimpan di variabel 'hapus\_barang'. Data barang yang dipilih akan ditampilkan, program meminta konfirmasi sebelum dihapus. Jika ya barang akan terhapus dan kembali ke *menu* admin. Memilih tidak berarti penghapusan dibatalkan dan kembali ke *menu* admin.

## K. Log-out dan Keluar Program

```

elif menu_admin == '5':
    os.system('cls || clear')
    konfirmasi1 = input('Yakin ingin log-out?(y/n): ')
    if konfirmasi1 == 'y':
        os.system('cls || clear')
        print('Kamu menekan tombol log-out!')
        break
    elif konfirmasi1 == 'n':
        print('Log-out dibatalkan')
    else:
        print('Pilihan tidak valid!')

elif menu_admin == '6':
    os.system('cls || clear')
    konfirmasi2 = input('Yakin ingin keluar?(y/n):')
    if konfirmasi2 == 'y':
        os.system('cls || clear')
        print('Anda menekan tombol keluar! Terima kasih telah

```

```

menggunakan program ini!')
    exit() #Keluar dari program
elif konfirmasi2 == 'n':
    print('Kembali ke program')
else:
    print('Pilihan tidak valid!')

else:
    os.system('cls || clear')
    print('Pilihan tidak valid!')

```

Gambar 3.11.1 Log-out dan Keluar

Pilihan ini ada untuk menambah interaksi dengan pengguna. Sebelum melakukan *log-out* dan *exit*, pengguna diminta mengkonfirmasi keputusan, jika *log-out* maka kembali ke *main menu* dan jika *exit* maka program berakhir.

## L. Menu Users

```

else: #Menu user
    print(f'Halo, {usr}!')
    print(''===MENU USER===')
1. Lihat barang
2. Log-out
3. Keluar'''
    menu_user = input('Pilih menu: ')
    if menu_user == '1':
        os.system('cls || clear')
        print('Daftar barang')

        tabel.clear_rows()

        for item in inventaris:
            tabel.add_row(item)
        print(tabel)
        banyak_barang = len(inventaris)
        print(f'Jumlah barang: {banyak_barang}')

    elif menu_user == '2':
        os.system('cls || clear')
        konfirmasi3 = input('Yakin ingin log-out?(y/n): ')
        if konfirmasi3 == 'y':
            os.system('cls || clear')
            print('Kamu menekan tombol log-out!')
            break
        elif konfirmasi3 == 'n':
            print('Log-out dibatalkan')

```

```

        else:
            print('Pilihan tidak valid!')

    elif menu_user == '3':
        os.system('cls || clear')
        konfirmasi4 = input('Yakin ingin keluar?(y/n):')
        if konfirmasi4 == 'y':
            os.system('cls || clear')
            print('Anda menekan tombol keluar! Terima kasih telah menggunakan program ini!')
            exit() #Keluar dari program
        elif konfirmasi4 == 'n':
            print('Kembali ke program')
        else:
            print('Pilihan tidak valid!')

    else:
        os.system('cls || clear')
        print('Pilihan tidak valid!')

```

Gambar 3.12.1

Menu *user* akan berjalan jika akun yang diregistrasikan sebelumnya *login*. *User* hanya bisa melihat daftar barang, sama seperti kode yang ada di *menu* admin, tabel daftar barang akan ditampilkan.

Pilihan lain ada *log-out* dan *exit* sama seperti sebelumnya, meminta konfirmasi sebelum memilih aksi.



#### 4. Hasil Output

```
===Menu===  
1. Login  
2. Register  
3. Keluar  
Pilih menu:1  
Login akun  
Username      :radja  
Password      :012
```

*Gambar 4.1 Login*

```
Username atau Password salah!  
Login akun  
Username      :
```

*Gambar 4.2 Username atau Password Salah*

```
Login berhasil!  
Halo, radja!  
===MENU ADMIN===  
1. Lihat barang  
2. Tambah barang  
3. Ubah data barang  
4. Hapus barang  
5. Log-out  
6. Keluar  
Pilih menu:
```

*Gambar 4.3 Login Sebagai radja dan Berhasil*

```
Registrasi
Username baru   :yusuf
Password baru   :010
```

*Gambar 4.4 Registrasi*

```
Login berhasil!
Halo, yusuf!
===MENU USER===
1. Lihat barang
2. Log-out
3. Keluar
Pilih menu: 
```

*Gambar 4.5 Login Sebagai User*

```
Registrasi
Username baru   :radja
Username sudah terpakai
Username baru    :asep
Password baru    :
Password minimal berjumlah 3 karakter!
Password baru    : 
```

*Gambar 4.6 Kesalahan saat Registrasi*

```

Tambah barang
Nama barang      :Keyboard
Contoh kode barang: BRG001, BRG002
Kode barang      :BRG006
Kode barang sudah digunakan! Gunakan kode lain
Contoh kode barang: BRG001, BRG002
Kode barang      :BRG008
Jumlah barang    :5
Kategori barang  :Elektronik
Kondisi barang   :Baik

```

Gambar 4.7 Proses Menambahkan Barang

```

Ubah jumlah barang:
Jumlah barang saat ini: 60
Pilih:
1. Tambah jumlah barang
2. Kurangi jumlah barang
Pilih (1/2): 2
Masukkan jumlah: 10

```

Gambar 4.8 Mengubah Jumlah Barang (Pulpen)

```

Daftar barang
+-----+-----+-----+-----+-----+
| Nama barang | Kode | Jumlah | Kategori | Kondisi |
+-----+-----+-----+-----+-----+
| Monitor     | BRG001 | 7      | Elektronik | Baik    |
| Meja kayu    | BRG002 | 10     | Perabot    | Baik    |
| Buku catatan | BRG003 | 30     | Alat tulis | Baik    |
| Kursi plastik | BRG004 | 20     | Perabot    | Baik    |
| Printer     | BRG005 | 5      | Elektronik | Baik    |
| Pulpen      | BRG006 | 50     | Alat tulis | Baik    |
| PC          | BRG007 | 7      | Elektronik | Baik    |
| Keyboard    | BRG008 | 5      | Elektronik | Baik    |
+-----+-----+-----+-----+-----+

Hapus barang
Masukkan kode barang yang ingin dihapus: BRG005

Data ditemukan:
Nama      : Printer
Kode      : BRG005
Jumlah    : 5
Kategori  : Elektronik
Kondisi   : Baik
Konfirmasi hapus barang?(y/n):y

```

Gambar 4.9 Menghapus Barang

Daftar barang					
Nama barang	Kode	Jumlah	Kategori	Kondisi	
Monitor	BRG001	7	Elektronik	Baik	
Meja kayu	BRG002	10	Perabot	Baik	
Buku catatan	BRG003	30	Alat tulis	Baik	
Kursi plastik	BRG004	20	Perabot	Baik	
Printer	BRG005	5	Elektronik	Baik	
Pulpen	BRG006	60	Alat tulis	Baik	
PC	BRG007	7	Elektronik	Baik	
Jumlah barang: 7					

Gambar 4.10 Tabel Sebelum CRUD

Daftar barang					
Nama barang	Kode	Jumlah	Kategori	Kondisi	
Monitor	BRG001	7	Elektronik	Baik	
Meja kayu	BRG002	10	Perabot	Baik	
Buku catatan	BRG003	30	Alat tulis	Baik	
Kursi plastik	BRG004	20	Perabot	Baik	
Pulpen	BRG006	50	Alat tulis	Baik	
PC	BRG007	7	Elektronik	Baik	
Keyboard	BRG008	5	Elektronik	Baik	
Jumlah barang: 7					

Gambar 4.11 Tabel Sesudah CRUD

## 5. Langkah-Langkah GIT

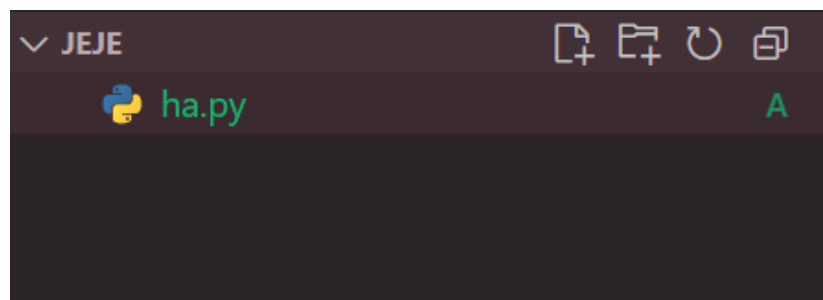
Jika dalam suatu folder kita sudah pernah melakukan *git init* dan *git remote* maka kita tidak perlu lagi melakukannya dan langsung ke *git add*.

### 5.1 GIT Add

```
PS D:\jeje> git add ha.py
PS D:\jeje> git add .
PS D:\jeje> 
```

Gambar 5.1 Proses Git Add

*Git add* berfungsi untuk memindah semua perubahan di area kerja ke indeks, ada dua cara untuk melakukan *git add*, yang pertama dengan mengetikkan nama *file* yang ingin di *add*. Jika kalian ingin menambahkan banyak *file* sekaligus, ketik “*git add .*”



Gambar 5.2 Sebuah File yang Telah di Add

*File* yang telah di *add* akan menampilkan huruf ‘A’ tepat di sebelah nama *file*, ini menandakan *file* sudah ditambahkan.

### 5.2 GIT Commit

```
PS D:\jeje> git commit -m "hellothere"
[main (root-commit) 55b666b] hellothere
1 file changed, 1 insertion(+)
create mode 100644 ha.py
PS D:\jeje> 
```

Gambar 5.3 Proses Git Commit

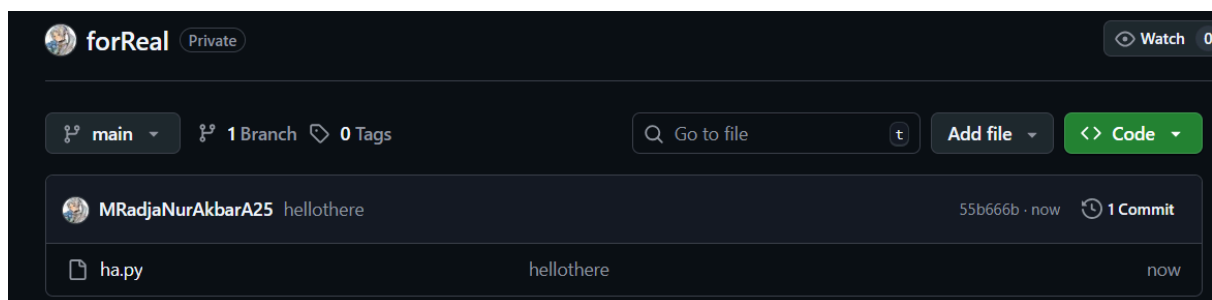
*Git commit* berfungsi untuk mengkonfirmasi setiap perubahan pada *repository* kalian dengan mengetikkan “git commit -m ‘pesan yang ingin ditulis’ ”

### 5.3 GIT Push

```
PS D:\jeje> git push -u origin main
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Writing objects: 100% (3/3), 235 bytes | 235.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/MRadjaNurAkbarA25/forReal.git
 * [new branch]      main -> main
branch 'main' set up to track 'origin/main'.
PS D:\jeje> █
```

*Gambar 5.4 Repository Lokal diunggah ke Github*

*Git push* berfungsi untuk mengunggah *file* lokal tadi ke GitHub kalian dengan mengetikkan “git push -u origin main”.



*Gambar 5.5 File sudah terunggah di Akun Github*