

**LAPORAN PRAKTIKUM
PEMROGRAMAN BERORIENTASI OBJEK (PBO) – [TUGAS BRP]**



Disusun Oleh

Andryano Shevchenko Limbong 123140205

**PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INFORMASI
INSTITUT TEKNOLOGI SUMATERA
2025**

Soal nomor 1

a. input soal

1. Menghitung Akar Kuadrat

Program ini meminta pengguna untuk memasukkan angka. Jika inputnya bukan angka, program akan menampilkan pesan "Input tidak valid. Harap masukkan angka yang valid." Jika angka yang dimasukkan negatif atau nol, program akan menampilkan pesan error yang relevan, seperti "Akar kuadrat dari nol tidak diperbolehkan." Jika inputnya valid (angka positif), program akan menghitung dan menampilkan akar kuadratnya. Terapkan exception dan error handling dalam menangani eror yang terjadi di dalam program.

contoh output:

Masukkan angka: Hello

Input tidak valid. Harap masukkan angka yang valid.

Masukkan angka: -5

Input tidak valid. Harap masukkan angka positif.

Masukkan angka: 0

Error: Akar kuadrat dari nol tidak diperbolehkan.

Masukkan angka: 25

Akar kuadrat dari 25 adalah 5.0

b. Penjelasan

Berikut adalah penjelasan dari kode tersebut:

1. Menggunakan Perulangan while True

Program terus berjalan sampai pengguna memasukkan angka yang valid.

```
import math

def main():
    while True:
        try:
            angka = input("Masukkan angka: ")
```

2. Menerima Input dari Pengguna

Pengguna diminta memasukkan angka. Input diubah menjadi tipe data float agar bisa menangani bilangan desimal.

```
try:
    angka = input("Masukkan angka: ")

    # Coba konversi input menjadi float
    angka = float(angka)
```

3. Memeriksa Validitas Input

Jika angka negatif, program menampilkan pesan bahwa input tidak valid. Jika angka nol, program menampilkan pesan error karena akar kuadrat nol dilarang dalam program ini. Jika angka positif, program menghitung akar kuadrat menggunakan `math.sqrt()` dan menampilkan hasilnya dengan dua angka di belakang koma (`:.2f`). Setelah berhasil menghitung akar kuadrat, program keluar dari loop menggunakan `break`.

```
# Cek apakah angka negatif atau nol
if angka < 0:
    print("Input tidak valid. Harap masukkan angka positif.")
elif angka == 0:
    print("Error: Akar kuadrat dari nol tidak diperbolehkan.")
else:
    hasil = math.sqrt(angka)
    print(f"Akar kuadrat dari {angka} adalah {hasil:.2f}")
    break # Keluar dari loop jika input valid
except ValueError:
```

4. Menangani Kesalahan Input

Jika pengguna memasukkan sesuatu yang bukan angka (misalnya huruf atau simbol), program akan menangkap `ValueError` dan meminta input ulang.

```
except ValueError:
    print("Input tidak valid. Harap masukkan angka yang valid.")

if __name__ == "__main__":
    main()
```

Kode ini memastikan bahwa pengguna hanya dapat memasukkan angka positif dan menghindari kesalahan yang mungkin terjadi saat memasukkan data yang tidak valid.

c. Source Code

```
import math

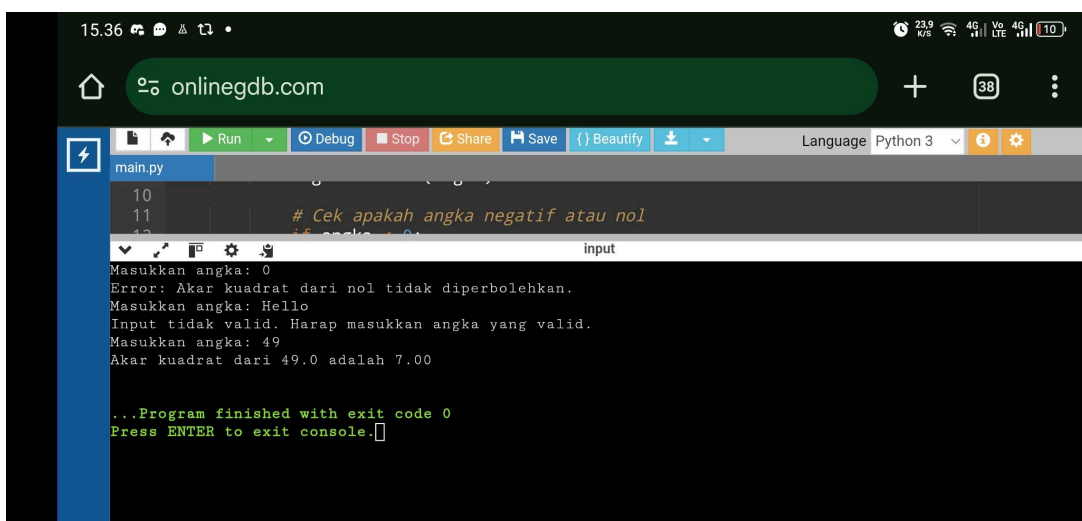
def main():
    while True:
        try:
            angka = input("Masukkan angka: ")

            # Coba konversi input menjadi float
            angka = float(angka)

            # Cek apakah angka negatif atau nol
            if angka < 0:
                print("Input tidak valid. Harap masukkan angka positif.")
            elif angka == 0:
                print("Error: Akar kuadrat dari nol tidak
diperbolehkan")
            else:
                hasil = math.sqrt(angka)
                print(f"Akar kuadrat dari {angka} adalah {hasil:.2f}")
                break # Keluar dari loop jika input valid
        except ValueError:
            print("Input tidak valid. Harap masukkan angka yang valid.")

if __name__ == "__main__":
    main()
```

d. Output Hasil



The screenshot shows a web-based Python IDE interface. At the top, there's a status bar with the time 15:36 and various system icons. Below that is a browser address bar showing 'onlinegdb.com'. The main area is divided into a code editor and a console. The code editor shows a Python script with line numbers 10, 11, and 12. The console shows the output of the program, including prompts for input and the resulting square root calculation.

```
main.py
10
11 # Cek apakah angka negatif atau nol
12 if angka < 0:

input
Masukkan angka: 0
Error: Akar kuadrat dari nol tidak diperbolehkan.
Masukkan angka: Hello
Input tidak valid. Harap masukkan angka yang valid.
Masukkan angka: 49
Akar kuadrat dari 49.0 adalah 7.00

...Program finished with exit code 0
Press ENTER to exit console
```

Soal Nomor 2:

a. Input Soal

2. Manajemen Daftar Tugas (To-Do List)

Program ini meminta pengguna untuk menambahkan, menghapus, dan menampilkan daftar tugas. Program ini menangani beberapa exception yang mungkin terjadi, seperti input yang tidak valid, mencoba menghapus tugas yang tidak ada, dan input kosong. Terapkan exception dan error handling, serta penerapan raising exception dalam menangani error yang terjadi di dalam program.

contoh output:

Pilih aksi:

1. Tambah tugas
2. Hapus tugas
3. Tampilkan daftar tugas
4. Keluar

Masukkan pilihan (1/2/3/4): 1

Masukkan tugas yang ingin ditambahkan: Belajar Python

Tugas berhasil ditambahkan!

Pilih aksi:

1. Tambah tugas
2. Hapus tugas
3. Tampilkan daftar tugas
4. Keluar

Masukkan pilihan (1/2/3/4): 3

Daftar Tugas:

- Belajar Python

Pilih aksi:

1. Tambah tugas
2. Hapus tugas
3. Tampilkan daftar tugas
4. Keluar

Masukkan pilihan (1/2/3/4): 2

Masukkan nomor tugas yang ingin dihapus: 2

Error: Tugas dengan nomor 2 tidak ditemukan.

Pilih aksi:

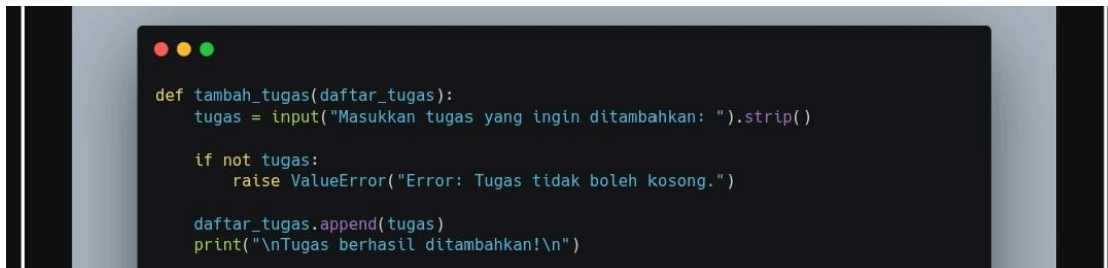
1. Tambah tugas
2. Hapus tugas
3. Tampilkan daftar tugas
4. Keluar

Masukkan pilihan (1/2/3/4): 4

Keluar dari program.

b. Penjelasan

1. Fungsi `tambah_tugas(daftar_tugas)` Meminta pengguna memasukkan tugas baru. Jika input kosong (`""`), program akan menampilkan error `ValueError`. Jika valid, tugas akan ditambahkan ke dalam `daftar_tugas`. Menampilkan pesan sukses setelah tugas ditambahkan.



```
def tambah_tugas(daftar_tugas):  
    tugas = input("Masukkan tugas yang ingin ditambahkan: ").strip()  
  
    if not tugas:  
        raise ValueError("Error: Tugas tidak boleh kosong.")  
  
    daftar_tugas.append(tugas)  
    print("\nTugas berhasil ditambahkan!\n")
```

2. Fungsi `hapus_tugas(daftar_tugas)` Memeriksa apakah daftar tugas kosong. Jika ya, program akan menampilkan error `IndexError`. Meminta pengguna memasukkan nomor tugas yang ingin dihapus. Memeriksa apakah nomor tugas valid (tidak kurang dari 1 dan tidak lebih dari jumlah tugas yang ada). Jika tidak valid, program akan menampilkan error `IndexError`. Jika valid, tugas dihapus dari `daftar_tugas` menggunakan `pop()`, dan program menampilkan pesan sukses. Jika input bukan angka, program menangkap `ValueError` dan meminta pengguna memasukkan angka yang valid.

```
def hapus_tugas(daftar_tugas):
    if not daftar_tugas:
        raise IndexError("Error: Daftar tugas kosong, tidak ada yang bisa dihapus.")

    try:
        nomor = int(input("Masukkan nomor tugas yang ingin dihapus: "))

        if nomor < 1 or nomor > len(daftar_tugas):
            raise IndexError(f"Error: Tugas dengan nomor {nomor} tidak ditemukan.")

        tugas_dihapus = daftar_tugas.pop(nomor - 1)
        print(f"\nTugas '{tugas_dihapus}' berhasil dihapus!\n")

    except ValueError:
        print("Error: Masukkan angka yang valid.\n")
```

3. Fungsi `tampilkan_tugas(daftar_tugas)` Jika daftar tugas kosong, program akan menampilkan pesan bahwa daftar kosong. Jika tidak, program mencetak semua tugas yang telah ditambahkan dengan format nomor urut.

```
def tampilkan_tugas(daftar_tugas):
    if not daftar_tugas:
        print("\nDaftar tugas kosong.\n")
    else:
        print("\nDaftar Tugas:")
        for i, tugas in enumerate(daftar_tugas, start=1):
            print(f"{i}. {tugas}")
        print()
```

4. Fungsi `main()` Membuat daftar kosong `daftar_tugas` untuk menyimpan tugas. Menjalankan `while True` untuk menampilkan menu utama dan meminta pengguna memilih aksi:
- Menjalankan fungsi `tambah_tugas()`
 - Menjalankan fungsi `hapus_tugas()`.
 - Menjalankan fungsi `tampilkan_tugas()`.
 - Menghentikan program dengan `break`.

Jika input tidak valid, program menampilkan error `ValueError`. Menggunakan `try-except` untuk menangani kesalahan agar program tidak crash jika terjadi input yang tidak valid.

```

def main():
    daftar_tugas = []

    while True:
        print("Pilih aksi:")
        print("1. Tambah tugas")
        print("2. Hapus tugas")
        print("3. Tampilkan daftar tugas")
        print("4. Keluar")

        pilihan = input("\nMasukkan pilihan (1/2/3/4): ").strip()

        try:
            if pilihan == "1":
                tambah_tugas(daftar_tugas)
            elif pilihan == "2":
                hapus_tugas(daftar_tugas)
            elif pilihan == "3":
                tampilkan_tugas(daftar_tugas)
            elif pilihan == "4":
                print("Keluar dari program.")
                break
            else:
                raise ValueError("Error: Pilihan tidak valid. Masukkan angka 1-4.")

        except Exception as e:
            print(e, "\n")

```

5. `if __name__ == "__main__":`: Memastikan bahwa program hanya berjalan jika file ini dieksekusi langsung, bukan diimpor sebagai modul.

```

if __name__ == "__main__":
    main()

```


c. Source Code

```
def tambah_tugas(daftar_tugas):
    tugas = input("Masukkan tugas yang ingin ditambahkan: ").strip()

    if not tugas:
        raise ValueError("Error: Tugas tidak boleh kosong.")

    daftar_tugas.append(tugas)
    print("\nTugas berhasil ditambahkan!\n")

def hapus_tugas(daftar_tugas):
    if not daftar_tugas:
        raise IndexError("Error: Daftar tugas kosong, tidak ada yang bisa dihapus.")
    try:
        nomor = int(input("Masukkan nomor tugas yang ingin dihapus: "))

        if nomor < 1 or nomor > len(daftar_tugas):
            raise IndexError(f"Error: Tugas dengan nomor {nomor} tidak ditemukan.")

        tugas_dihapus = daftar_tugas.pop(nomor - 1)
        print(f"\nTugas '{tugas_dihapus}' berhasil dihapus!\n")

    except ValueError:
        print("Error: Masukkan angka yang valid.\n")

def tampilkan_tugas(daftar_tugas):
    if not daftar_tugas:
        print("\nDaftar tugas kosong.\n")
    else:
        print("\nDaftar Tugas:")
        for i, tugas in enumerate(daftar_tugas, start=1):
            print(f"{i}. {tugas}")
        print()

def main():
    daftar_tugas = []

    while True:
        print("Pilih aksi:")
        print("1. Tambah tugas")
        print("2. Hapus tugas")
        print("3. Tampilkan daftar tugas")
        print("4. Keluar")

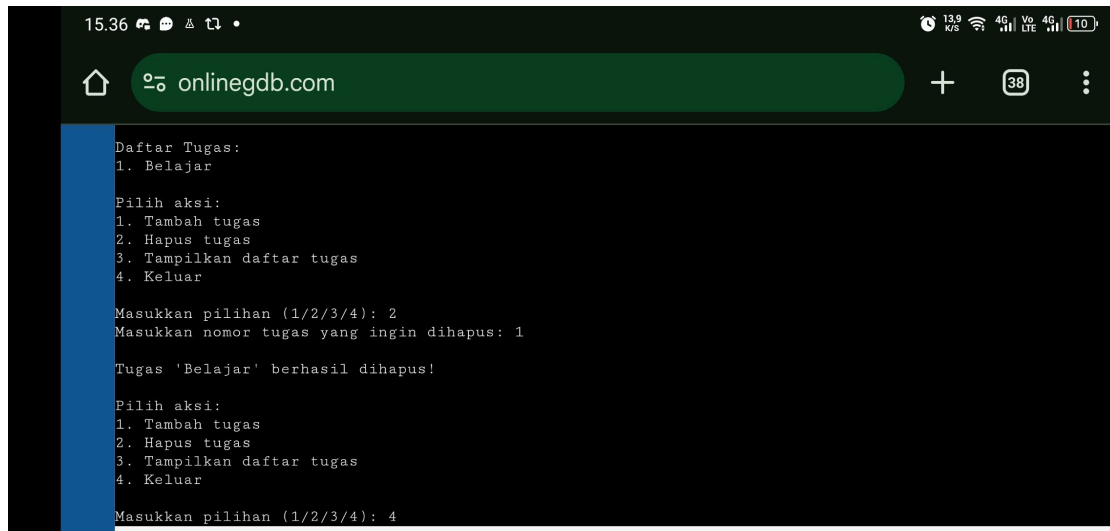
        pilihan = input("\nMasukkan pilihan (1/2/3/4): ").strip()

        try:
            if pilihan == "1":
                tambah_tugas(daftar_tugas)
            elif pilihan == "2":
                hapus_tugas(daftar_tugas)
            elif pilihan == "3":
                tampilkan_tugas(daftar_tugas)
            elif pilihan == "4":
                print("Keluar dari program.")
                break
            else:
                raise ValueError("Error: Pilihan tidak valid. Masukkan angka 1-4.")

        except Exception as e:
            print(e, "\n")

if __name__ == "__main__":
    main()
```

d. Hasil Output



```
15.36 13.9 k/s 4G Vo LTE 10
onlinegdb.com
Daftar Tugas:
1. Belajar

Pilih aksi:
1. Tambah tugas
2. Hapus tugas
3. Tampilkan daftar tugas
4. Keluar

Masukkan pilihan (1/2/3/4): 2
Masukkan nomor tugas yang ingin dihapus: 1

Tugas 'Belajar' berhasil dihapus!

Pilih aksi:
1. Tambah tugas
2. Hapus tugas
3. Tampilkan daftar tugas
4. Keluar

Masukkan pilihan (1/2/3/4): 4
```

Soal nomor 3

a. Input soal

Sistem Manajemen Hewan (Zoo Management System)

Pada kasus ini, kalian akan membuat sebuah sistem untuk mengelola berbagai jenis hewan di kebun binatang.

Setiap hewan memiliki karakteristik dan perilaku yang berbeda, dan sistem ini akan mengimplementasikan konsep-konsep OOP yang telah dipelajari sebelumnya beserta error handling yang kita pelajari hari ini.

Deskripsi Program:

Polimorfisme: Setiap hewan dapat memiliki metode `make_sound()` yang berbeda-beda sesuai dengan jenis hewan tersebut. Metode ini akan dipanggil melalui objek hewan yang berbeda jenis (misalnya, anjing, kucing, dll), kalo mau tambah metode lain boleh.

Abstraksi: Kalian akan membuat kelas abstrak `Animal` yang mendefinisikan metode `make_sound()` yang harus diimplementasikan oleh semua kelas turunan.

Enkapsulasi: Data terkait hewan akan disembunyikan di dalam kelas dan hanya bisa diakses atau dimodifikasi melalui metode tertentu.

Inheritance: Semua hewan akan mewarisi kelas dasar `Animal` dan dapat menambahkan perilaku atau atribut mereka sendiri.

Exception Handling: Program akan menangani error jika ada input yang tidak valid, misalnya, saat menambahkan hewan tanpa nama atau usia yang benar.

refreshing otak 4 pilar OOP:

Encapsulation: semua properti harus bersifat private dan hanya bisa diakses lewat method, getter untuk mengambil dan setter untuk merubah nilainya.

Inheritance: suatu class menjadi subclass dari superclass yang mana subclass mempunyai properti dan method yang sama dengan superclass.

Abstraction: Membuat class yang mana class tersebut merupakan blue print dari kelas lain, kelas yang untuk blueprint class lain tidak bisa dijadikan blueprint objek, dan juga akan memaksa class blueprint untuk memakai abstractmethod yang ada di dalam kelas abstrak(wajib dipake semuanya), abstractmethod di kelas abstract tidak perlu diimplementasikan di kelas abstract(isi 'pass' aja)

Polimorphism: Mengoverride suatu fungsi di dalam kelas induk di kelas anaknya

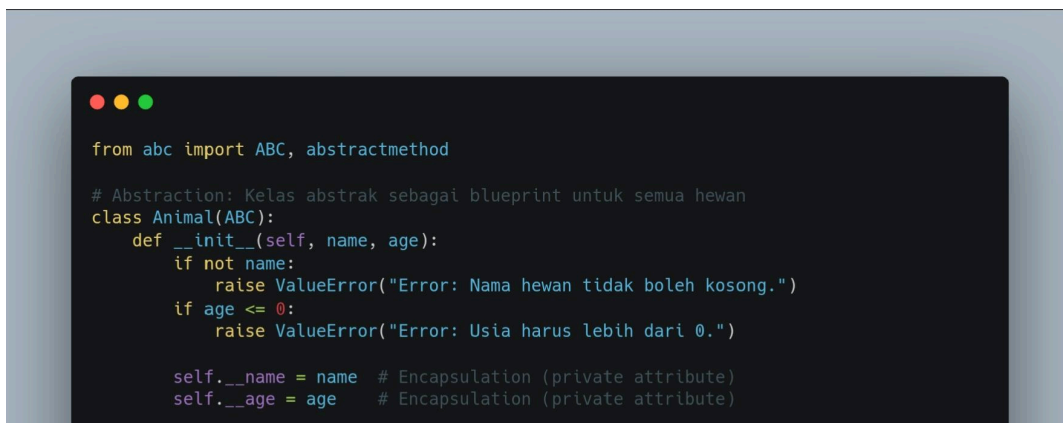
Buat sesuai imajinasi kalian saja, yang penting sesuai deskripsi..

b. Penjelasan

1. kelas Animal (Superclass / Abstraksi)

Kelas ini berfungsi sebagai blueprint untuk semua hewan. `__init__(self, name, age)`

- Fungsi: Konstruktor untuk inisialisasi nama dan usia hewan.
- Validasi: Jika nama kosong → `ValueError`, Jika usia ≤ 0 → `ValueError`
- Encapsulation: Menggunakan atribut private `__name` dan `__age`.



```
from abc import ABC, abstractmethod

# Abstraction: Kelas abstrak sebagai blueprint untuk semua hewan
class Animal(ABC):
    def __init__(self, name, age):
        if not name:
            raise ValueError("Error: Nama hewan tidak boleh kosong.")
        if age <= 0:
            raise ValueError("Error: Usia harus lebih dari 0.")

        self.__name = name # Encapsulation (private attribute)
        self.__age = age   # Encapsulation (private attribute)
```

2. make_sound(self) (Abstrak Method)

- Fungsi: Metode abstrak yang wajib diimplementasikan di setiap subclass.
- Menggunakan @abstractmethod, jadi kelas Animal tidak bisa dibuat objeknya langsung.

```
@abstractmethod
def make_sound(self):
    pass # Wajib diimplementasikan di subclass

# Getter
```

3. Getter & Setter

- Fungsi: Untuk mengakses (getter) dan mengubah (setter) atribut private __name dan __age.
- Validasi: Jika nama kosong → ValueError, Jika usia ≤ 0 → ValueError

```
# Getter
def get_name(self):
    return self.__name

def get_age(self):
    return self.__age

# Setter (hanya jika ingin mengubah data)
def set_name(self, name):
    if not name:
        raise ValueError("Error: Nama tidak boleh kosong.")
    self.__name = name

def set_age(self, age):
    if age <= 0:
        raise ValueError("Error: Usia harus lebih dari 0.")
    self.__age = age
```

4. info(self) (Polimorfisme)

Fungsi: Mengembalikan string informasi nama dan usia hewan. Dapat di-override di subclass.

```
# Polimorfisme: Metode yang bisa di-override
def info(self):
    return f"{self.__name} (Usia: {self.__age} tahun)"

# Inheritance: Hewan-hewan spesifik mewarisi kelas Animal
```

5. Kelas Dog, Cat, Lion, Elephant (Inheritance)

Fungsi: Subclass yang mewarisi Animal dan harus mengimplementasikan make_sound(). Setiap subclass memiliki suara unik.

```
# Inheritance: Hewan-hewan spesifik mewarisi kelas Animal
class Dog(Animal):
    def make_sound(self):
        return "Woof! Woof!"

class Cat(Animal):
    def make_sound(self):
        return "Meow! Meow!"

class Lion(Animal):
    def make_sound(self):
        return "Roarrr!"

class Elephant(Animal):
    def make_sound(self):
        return "Trumpettt!"
```

6. Kelas Zoo (Manajemen Kebun Binatang)

Kelas ini mengelola daftar hewan di kebun binatang. `__init__(self)`

Fungsi: Konstruktor untuk inisialisasi daftar hewan (`__animals`, private).

```
# Kelas untuk mengelola kebun binatang
class Zoo:
    def __init__(self):
        self.__animals = [] # Private list untuk menyimpan hewan

    def add_animal(self, animal):
```

`add_animal(self, animal)`

- Fungsi: Menambahkan hewan ke dalam daftar kebun binatang.
- Validasi: Jika animal bukan turunan Animal → TypeError.

```
def add_animal(self, animal):
    if not isinstance(animal, Animal):
        raise TypeError("Error: Objek yang ditambahkan harus turunan dari
Animal.")
    self.__animals.append(animal)
    print(f"{animal.get_name()} berhasil ditambahkan ke kebun binatang!")
```

`show_animals(self)`

Fungsi: Menampilkan semua hewan beserta suaranya. Jika daftar kosong, tampilkan pesan "Kebun binatang kosong".

```
def show_animals(self):
    if not self.__animals:
        print("Kebun binatang kosong.")
    else:
        print("\nDaftar Hewan di Kebun Binatang:")
        for idx, animal in enumerate(self.__animals, start=1):
            print(f"{idx}. {animal.info()} - Suara: {animal.make_sound()}")
```

remove_animal(self, index)

- Fungsi: Menghapus hewan berdasarkan nomor.
- Validasi: Jika input bukan angka → ValueError

Jika nomor tidak ada → IndexError

```
def remove_animal(self, index):
    try:
        index = int(index) - 1
        if index < 0 or index >= len(self.__animals):
            raise IndexError("Error: Hewan dengan nomor tersebut tidak ditemukan.")
        removed_animal = self.__animals.pop(index)
        print(f"{removed_animal.get_name()} telah dihapus dari kebun binatang.")
    except ValueError:
        print("Error: Masukkan nomor yang valid.")
    except IndexError as e:
        print(e)
```

Fungsi main()

Fungsi utama yang mengelola menu interaktif untuk pengguna. main()

1. Membuat objek Zoo().
2. Menampilkan menu pilihan:
 - 1: Tambah hewan
 - 2: Tampilkan daftar hewan
 - 3: Hapus hewan
 - 4: Keluar

3. Validasi input dengan try-except untuk menangani error.

```

# Main program
def main():
    zoo = Zoo()

    while True:
        print("\n=== Sistem Manajemen Kebun Binatang ===")
        print("1. Tambah Hewan")
        print("2. Tampilkan Hewan")
        print("3. Hapus Hewan")
        print("4. Keluar")

        pilihan = input("\nMasukkan pilihan (1/2/3/4): ").strip()

        try:
            if pilihan == "1":
                print("\nJenis Hewan: 1. Anjing 2. Kucing 3. Singa 4. Gajah")
                jenis = input("Pilih jenis hewan (1/2/3/4): ").strip()

                nama = input("Masukkan nama hewan: ").strip()
                usia = int(input("Masukkan usia hewan: ").strip())

                if jenis == "1":
                    hewan = Dog(nama, usia)
                elif jenis == "2":
                    hewan = Cat(nama, usia)
                elif jenis == "3":
                    hewan = Lion(nama, usia)
                elif jenis == "4":
                    hewan = Elephant(nama, usia)
                else:
                    raise ValueError("Error: Pilihan jenis hewan tidak valid.")

                zoo.add_animal(hewan)

            elif pilihan == "2":
                zoo.show_animals()

            elif pilihan == "3":
                nomor = input("Masukkan nomor hewan yang ingin dihapus: ").strip()
                zoo.remove_animal(nomor)

            elif pilihan == "4":
                print("Keluar dari program. Sampai jumpa!")
                break

            else:
                raise ValueError("Error: Pilihan tidak valid. Masukkan angka 1-4.")

        except Exception as e:
            print(e)

if __name__ == "__main__":
    main()

```

c. Source Code

```
from abc import ABC, abstractmethod

# Abstraction: Kelas abstrak sebagai blueprint untuk semua hewan
class Animal(ABC):
    def __init__(self, name, age):
        if not name:
            raise ValueError("Error: Nama hewan tidak boleh kosong.")
        if age <= 0:
            raise ValueError("Error: Usia harus lebih dari 0.")

        self.__name = name # Encapsulation (private attribute)
        self.__age = age   # Encapsulation (private attribute)

    @abstractmethod
    def make_sound(self):
        pass # Wajib diimplementasikan di subclass

    # Getter
    def get_name(self):
        return self.__name

    def get_age(self):
        return self.__age

    # Setter (hanya jika ingin mengubah data)
    def set_name(self, name):
        if not name:
            raise ValueError("Error: Nama tidak boleh kosong.")
        self.__name = name

    def set_age(self, age):
        if age <= 0:
            raise ValueError("Error: Usia harus lebih dari 0.")
        self.__age = age

    # Polimorfisme: Metode yang bisa di-override
    def info(self):
        return f"{self.__name} (Usia: {self.__age} tahun)"

# Inheritance: Hewan-hewan spesifik mewarisi kelas Animal
class Dog(Animal):
    def make_sound(self):
        return "Woof! Woof!"

class Cat(Animal):
    def make_sound(self):
        return "Meow! Meow!"

class Lion(Animal):
    def make_sound(self):
        return "Roarr!"

class Elephant(Animal):
    def make_sound(self):
        return "Trumpettt!"
```



```

# Kelas untuk mengelola kebun binatang
class Zoo:
    def __init__(self):
        self.__animals = [] # Private list untuk menyimpan hewan

    def add_animal(self, animal):
        if not isinstance(animal, Animal):
            raise TypeError("Error: Objek yang ditambahkan harus turunan dari
Animal.")
        self.__animals.append(animal)
        print(f"{animal.get_name()} berhasil ditambahkan ke kebun binatang!")

    def show_animals(self):
        if not self.__animals:
            print("Kebun binatang kosong.")
        else:
            print("\nDaftar Hewan di Kebun Binatang:")
            for idx, animal in enumerate(self.__animals, start=1):
                print(f"{idx}. {animal.info()} - Suara: {animal.make_sound()}")

    def remove_animal(self, index):
        try:
            index = int(index) - 1
            if index < 0 or index >= len(self.__animals):
                raise IndexError("Error: Hewan dengan nomor tersebut tidak
ditemukan.")
            removed_animal = self.__animals.pop(index)
            print(f"{removed_animal.get_name()} telah dihapus dari kebun binatang.")
        except ValueError:
            print("Error: Masukkan nomor yang valid.")
        except IndexError as e:
            print(e)

# Main program
def main():
    zoo = Zoo()

    while True:
        print("\n=== Sistem Manajemen Kebun Binatang ===")
        print("1. Tambah Hewan")
        print("2. Tampilkan Hewan")
        print("3. Hapus Hewan")
        print("4. Keluar")

        pilihan = input("\nMasukkan pilihan (1/2/3/4): ").strip()

        try:
            if pilihan == "1":
                print("\nJenis Hewan: 1. Anjing 2. Kucing 3. Singa 4. Gajah")
                jenis = input("Pilih jenis hewan (1/2/3/4): ").strip()

                nama = input("Masukkan nama hewan: ").strip()
                usia = int(input("Masukkan usia hewan: ").strip())

                if jenis == "1":
                    hewan = Dog(nama, usia)
                elif jenis == "2":
                    hewan = Cat(nama, usia)
                elif jenis == "3":
                    hewan = Lion(nama, usia)
                elif jenis == "4":
                    hewan = Elephant(nama, usia)
                else:
                    raise ValueError("Error: Pilihan jenis hewan tidak valid.")

                zoo.add_animal(hewan)

            elif pilihan == "2":
                zoo.show_animals()

            elif pilihan == "3":
                nomor = input("Masukkan nomor hewan yang ingin dihapus: ").strip()
                zoo.remove_animal(nomor)

            elif pilihan == "4":
                print("Keluar dari program. Sampai jumpa!")
                break

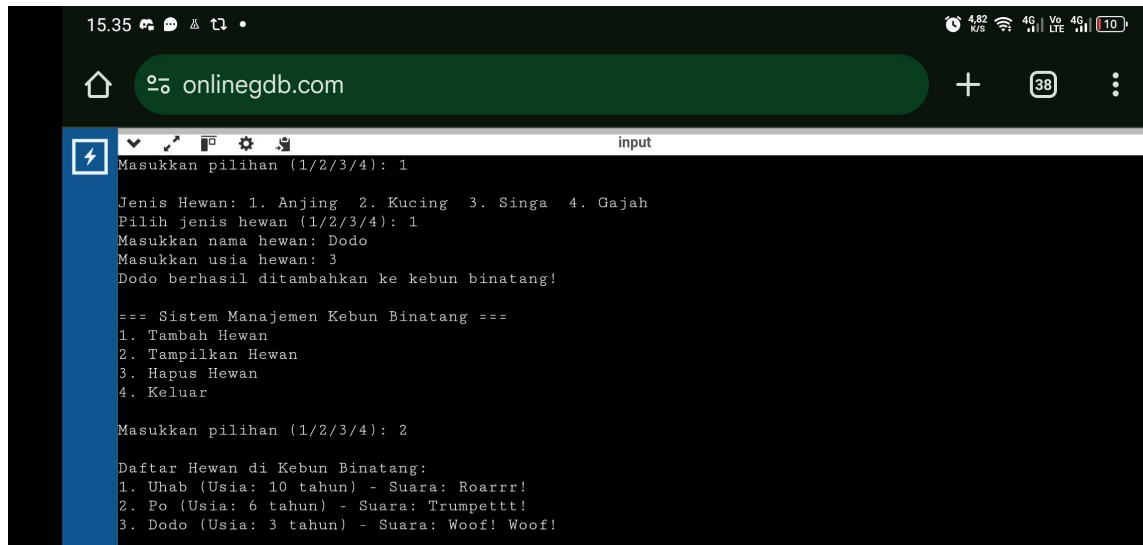
            else:
                raise ValueError("Error: Pilihan tidak valid. Masukkan angka 1-4.")

        except Exception as e:
            print(e)

if __name__ == "__main__":
    main()

```

d. Hasil



```
15:35 4.82 4G VoLTE 10
onlinegdb.com
input
Masukkan pilihan (1/2/3/4): 1
Jenis Hewan: 1. Anjing 2. Kucing 3. Singa 4. Gajah
Pilih jenis hewan (1/2/3/4): 1
Masukkan nama hewan: Dodo
Masukkan usia hewan: 3
Dodo berhasil ditambahkan ke kebun binatang!

=== Sistem Manajemen Kebun Binatang ===
1. Tambah Hewan
2. Tampilkan Hewan
3. Hapus Hewan
4. Keluar

Masukkan pilihan (1/2/3/4): 2

Daftar Hewan di Kebun Binatang:
1. Uhab (Usia: 10 tahun) - Suara: Roarrrr!
2. Po (Usia: 6 tahun) - Suara: Trumpettt!
3. Dodo (Usia: 3 tahun) - Suara: Woof! Woof!
```

Lampiran

1. [Link Percakapan LLM:](https://chatgpt.com/share/67e294a7-f724-8012-9347-ed0f25d4f1a5)
<https://chatgpt.com/share/67e294a7-f724-8012-9347-ed0f25d4f1a5>

(Jika menggunakan LLM atau Referensi website dalam pembuatan laporan, baik untuk generate code ataupun penulisan text silahkan sertakan dokumentasinya bisa berupa link ataupun screenshot percakapan. Jika tidak melampirkan dan ketahuan menggunakan nilai = 0)

Contoh :

1. [Link Percakapan LLM](#)
2. [Web Referensi - DuniaIlkom](#)