**LAPORAN AKHIR**

**PRAKTIKUM PEMROGRAMAN BERBASIS OBJEK**



**Disusun Oleh :**

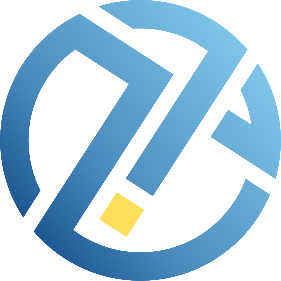
**NAMA : ADYTTA PUTRA TARIGAN**

**N.R.P : 240441100139**

**KELAS : 2A**

**DOSEN :** **YUDHA DWI PUTRA NEGARA, S.Kom., M.Kom.**

**ASISTEN : AHMAD RIKHAN ARBA’I**



**LABORATORIUM BISNIS INTELIJEN SISTEM**

**PRODI SISTEM INFORMASI**

**JURUSAN TEKNIK INFORMATIKA**

**FAKULTAS TEKNIK**

**UNIVERSITAS TRUNOJOYO MADURA**

**LAPORAN AKHIR**

**PRAKTIKUM PEMROGRAMAN BERBASIS OBJEK**

**Disusun Oleh :**

**NAMA : ADYTTA PUTRA TARIGAN**

**N.R.P : 240441100139**

**KELAS : 2A**

**DOSEN : YUDHA DWI PUTRA NEGARA, S.Kom., M.Kom.**

**ASISTEN : AHMAD RIKHAN ARBAI’I**

**TGL PRAKTIKUM :**

**Disetujui :**

**Asisten**

**AHMAD RIKHAN ARBA’I**

**23.04.411.00060**

**LABORATORIUM BISNIS INTELIJEN SISTEM**

**PRODI SISTEM INFORMASI**

**JURUSAN TEKNIK INFORMATIKA**

**FAKULTAS TEKNIK**

**UNIVERSITAS TRUNOJOYO MADURA**

**LAPORAN RESMI**

# MODUL I (*CLASS*, *OBJECT* DAN *CONSTRUCTOR*)

**PEMROGRAMAN BERBASIS OBJEK**

**NAMA : ADYTTA PUTRA TARIGAN**

**N.R.P : 240441100139**

**DOSEN : YUDHA DWI PUTRA NEGARA, S.KOM., M.KOM.**

**ASISTEN :** **AHMAD RIKHAN ARBA’I**

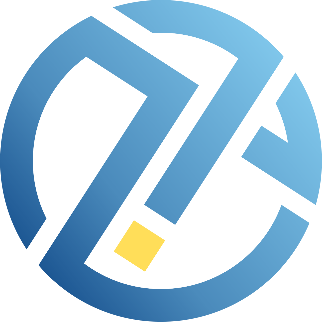
**TGL PRAKTIKUM : 22 MARET 2025**

**Disetujui : 10 MARET 2025**

**Asisten**

**AHMAD RIKHAN ARBA’I**

**23.04.411.00192**



**LABORATORIUM TEKNOLOGI INFORMASI**

**PRODI SISTEM INFORMASI**

**JURUSAN TEKNIK INFORMATIKA**

**FAKULTAS TEKNIK**

**UNIVERSITAS TRUNOJOYO MADURA**

## BAB I PENDAHULUAN

### Latar Belakang

Dalam dunia pemrograman modern, konsep *Class*, *Object*, dan *Constructor* menjadi fondasi utama dalam paradigma Pemrograman Berorientasi Objek (OOP). Dengan adanya *class*, programmer dapat mendefinisikan blueprint atau cetak biru dari suatu objek, yang berisi atribut dan metode untuk mengelola data serta perilaku objek tersebut. *Object* adalah instansi dari *class* yang dapat digunakan untuk merepresentasikan entitas nyata dalam dunia nyata, seperti mobil, mahasiswa, atau akun pengguna. Konsep ini memungkinkan pengembangan perangkat lunak yang lebih terstruktur, modular, dan mudah dikelola.

Salah satu elemen penting dalam OOP adalah *constructor*, yaitu metode khusus yang dieksekusi saat objek dibuat. *Constructor* digunakan untuk menginisialisasi nilai awal dari atribut suatu objek, sehingga tidak perlu diatur secara manual setelah objek dibuat. Dengan adanya *constructor*, kode program menjadi lebih efisien dan mudah dibaca. Selain itu, *constructor* juga memungkinkan pembuatan objek yang konsisten dan bebas dari kesalahan pengaturan awal.

Pemahaman yang baik tentang *class*, *object*, dan *constructor* sangat penting bagi programmer untuk mengembangkan aplikasi yang skalabel dan mudah diperbaiki. Dengan menerapkan prinsip OOP secara tepat, pengembangan perangkat lunak menjadi lebih fleksibel, mendukung penggunaan kembali kode (code reusability), serta memudahkan proses debugging dan pemeliharaan jangka panjang.

### Tujuan

* Mahasiswa mampu memahami konsep *Class*
* Mahasiswa mampu menggunakan *Object* dalam program
* Mahasiswa mampu mengimplementasikan *Constructor* secara efektif

## BAB II DASAR TEORI

### *Class*, *Object* dan *Constructor*

#### *Class*

*Class* adalah blueprint atau cetak biru yang digunakan untuk mendefinisikan struktur dan perilaku dari suatu objek dalam pemrograman berorientasi objek (OOP). *Class* berisi atribut (variabel) dan *method* (fungsi) yang menjelaskan karakteristik dan aksi yang dapat dilakukan oleh objek yang dibuat dari *class* tersebut.

Contoh Analogis, Bayangkan sebuah *class* sebagai cetak biru atau blueprint adalah rumah. Dalam blueprint ini, kita menentukan apa saja yang akan dimiliki oleh rumah, seperti jumlah kamar, warna tembok, dll. Namun, blueprint itu sendiri bukanlah rumah, melainkan hanya rancangan atau panduan untuk membangun rumah.

1. Ciri-Ciri *Class*.

a. Mendefinisikan atribut (properti) dan *method* (fungsi) yang bisa dimiliki oleh objek.

b. Menjadi cetak biru blueprint untuk membuat objek.

c. Penulisan Nama pada *class* diawali dengan huruf kapital sesuai konvensi pemrograman Python.

2. Syntax Penulisan pada *Class*

Dalam Python, *Class* merupakan blueprint atau cetak biru untuk membuat objek. *Class* mendefiniskan atribut (variabel yang dimiliki oleh objek) dan metode (fungsi yang dapat dilakukan oleh objek).



#### *Object*

*Object* adalah instansiasi atau perwujudan konkret dari suatu *Class*. Ketika kita membuat *Object*, kita menggunakan struktur yang sudah didefinisikan oleh *Class* dan memberikan nilai nyata untuk atribut-atribut tersebut. Objek dapat memanfaatkan *method* yang ada di *class* untuk melakukan aksi atau manipulasi data.

Contoh Analogis. Jika *Class* adalah cetak biru atau blueprint rumah, maka *Object* adalah rumah yang sebenarnya dibangun berdasarkan cetak biru tersebut. Setiap rumah yang dibangun dari blueprint yang sama bisa memiliki detail yang berbeda (misalnya warna yang berbeda, jumlah kamar yang berbeda).

1. Ciri-Ciri *Object*:

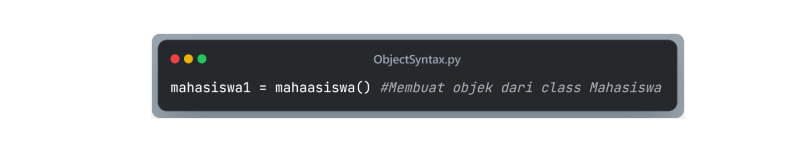
a. Dibuat menggunakan *class*.

b. Memiliki nilai konkret untuk atribut yang didefinisikan oleh *class*.

c. Dapat memanggil *method*dari *class* untuk berinteraksi atau memanipulasi atributnya.

2. Syntax Penulisan pada *Object*.

*Object* merupakan instansi (instance) dari sebuah kelas. Setiap objek memiliki nilai atribut yang unik.



#### *Constructor*

*Constructor* adalah *method* khusus yang digunakan untuk menginisialisasi

atribut dari *Object* saat objek tersebut dibuat. Dalam Python, *constructor* ditandai dengan *method* init (). Fungsi ini akan dijalankan secara otomatis saat kita membuat objek dari *class*. Biasanya, *constructor* digunakan untuk menetapkan nilai awal dari atribut-atribut objek.

Contoh Analogis: *Constructor* adalah pekerja yang menyiapkan rumah saat dibangun dari blueprint (*class*). Pekerja ini akan menyiapkan semua hal yang diperlukan (seperti memasang pintu, jendela, dll.) agar rumah siap digunakan.

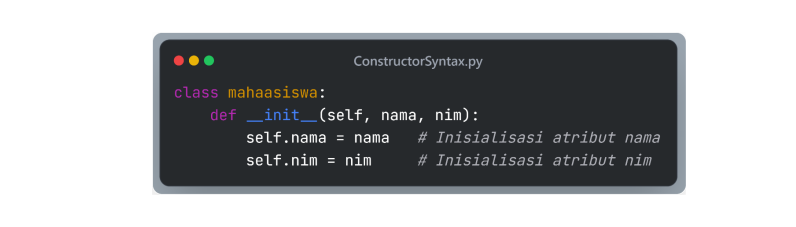
1. Ciri-Ciri *Constructor*

a. Didefinisikan menggunakan init (self, ...).

b. Secara otomatis dipanggil saat objek dibuat.

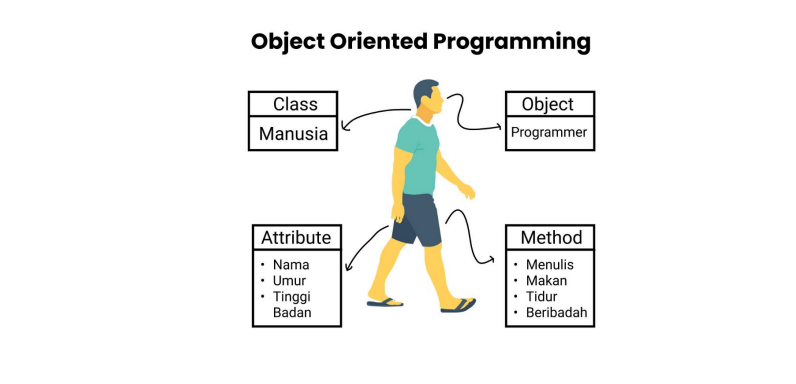
c. Biasanya digunakan untuk menginisialisasi atribut objek dengan nilai yang diberikan saat pembuatan objek.

2. Syntax Penulisan pada *Constructor*

*Constructor* merupakan metode khusus dalam sebuah kelas yang secara otomatis dipanggil ketika sebuah objek dibuat. *Constructor* digunakan untuk menginisialisasi atribut-atribut objek.

#### Ilustrasi *Class*, *Object*, dan *Constructor*

Ilustrasi gambar yang memberikan representasi visual sederhana namun efektif tentang konsep dasar kelas, objek, dan konstruktor dalam OOP Python. Dengan memahami konsep-konsep ini, Anda dapat mulai membangun aplikasi yang lebih kompleks dan terstruktur dengan baik menggunakan bahasa pemrograman Python.



Gambar di atas menjelaskan konsep *Object* Oriented Programming (OOP) atau Pemrograman Berorientasi Objek dengan menggunakan contoh manusia. Berikut adalah penjelasan dari elemen-elemen yang ada dalam gambar:

1. *Class*(kelas) “Manusia”:

Di bagian kiri atas gambar terdapat sebuah kotak dengan label *Class*

yang berisi kata "Manusia". Ini menunjukkan bahwa dalam konsep OOP, Manusia adalah kelas yang menjadi cetak biru bagi berbagai objek yang dapat dibuat darinya.

1. *Object* (Objek) - "Programmer"

Di bagian kanan atas gambar, terdapat kotak dengan label *Object* yang berisi kata "Programmer". Ini menunjukkan bahwa Programmer adalah salah satu contoh objek yang dibuat dari kelas Manusia. Dalam dunia nyata, manusia bisa memiliki berbagai profesi, dan salah satu perwujudan dari manusia adalah seorang programmer.

1. Attribute (Atribut) :

Di bagian kiri bawah gambar, terdapat kotak dengan label Attribute yang berisi daftar karakteristik. Atribut adalah properti atau karakteristik yang dimiliki oleh sebuah objek.

Contoh atribut dalam kelas manusia :

1. Nama
2. Umur
3. Tinggi badan
4. *Method* (Metode) :

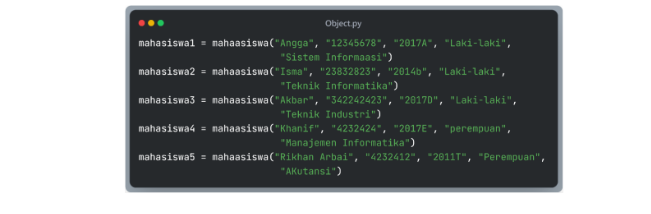
Di bagian kanan bawah gambar, terdapat kotak dengan label *Method* yang berisi daftar perilaku atau tindakan yang dapat dilakukan oleh objek.

Contoh metode dalam kelas manusia :

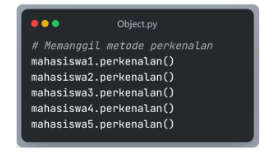
1. Menulis
2. Makan
3. Tidur
4. Beribadah

#### Program Python Dalam Penerapan *Class, Object*, dan *Constructor*

Berikut merupakan program Perkenalan Mahasiswa menggunakan *Class* dengan bahasa Python.

Berikut merupakan program Perkenalan Mahasiswa menggunakan *Object* dengan bahas Python.

Berikut merupakan program Perkenalan Mahasiswa menggunakan *Constructor* dengan bahas Python.



## BAB III TUGAS PENDAHULUAN

## BAB IV IMPLEMENTASI

### Tugas Praktikum

#### Tugas Praktikum No. 1

Buatlah *class* manusia dengan dengan attribut nama, umur dan alamat serta memiliki *method* berjalan dan berlari. Buatlah (minimal 5) objek dari *class* tersebut.

#### Tugas Praktikum No. 2

Buatlah program untuk menampilkan nama, nim, jurusan/prodi, dan alamat mahasiswa menggunakan konsep OOP. Nama, nim, jurusan/prodi, dan alamat bersifat dinamis, dengan kata lain menggunakan input dari pengguna. Buat minimal 3 objek berdasarkan input pengguna.

#### Tugas Praktikum No. 3

Buatlah tiga *class* hewan dengan attribut yang dimiliki hewan tersebut beserta *method*-*method*nya juga. Buatlah beberapa objek dari *class*-*class* tersebut (minimal 3) menggunakan looping.

### Source Code

#### Source Code Soal 1

|  |
| --- |
| ***class* manusia:**  **def \_\_init\_\_(self, nama, umur, alamat):**  **self.nama = nama**  **self.umur = umur**  **self.alamat = alamat**  **def berjalan(self):**  **print(f"{self.nama} sedang berjalan.")**  **def berlari(self):**  **print(f"{self.nama} sedang berlari.")**  **orang1 = manusia("Adytta", 19, "Jakarta")**  **orang2 = manusia("Fauzi", 20, "Jakarta")**  **orang3 = manusia("Abdul", 19, "Jakarta")**  **orang4 = manusia("Ilham", 20, "Jakarta")**  **orang5 = manusia("Teddy", 19, "Jakarta")**  **orang6 = manusia("Rehan", 20, "Jakarta")**  **orang1.berjalan()**  **orang2.berlari()**  **orang3.berjalan()**  **orang4.berlari()**  **orang5.berjalan()**  **orang6.berlari()** |

#### Source Code Soal 2

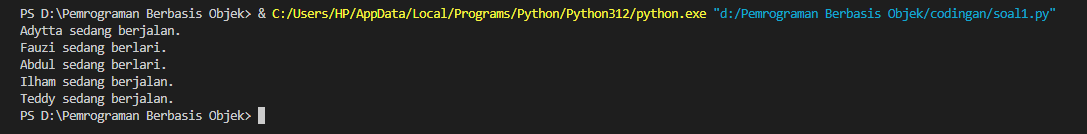
|  |
| --- |
| ***class* Mahasiswa:**  **def \_\_init\_\_(self, nama, nim, jurusan, alamat):**  **self.nama = nama**  **self.nim = nim**  **self.jurusan = jurusan**  **self.alamat = alamat**  **def tampilkan\_info(self):**  **print("\nData Mahasiswa")**  **print(f"Nama      : {self.nama}")**  **print(f"NIM       : {self.nim}")**  **print(f"Jurusan   : {self.jurusan}")**  **print(f"Alamat    : {self.alamat}")**  **mahasiswa\_list = []**  **for i in range(3):**  **print(f"\nMasukkan data mahasiswa ke-{i+1}:")**  **nama = input("Nama: ")**  **nim = input("NIM: ")**  **jurusan = input("Jurusan/Prodi: ")**  **alamat = input("Alamat: ")**  **mahasiswa = Mahasiswa(nama, nim, jurusan, alamat)**  **mahasiswa\_list.append(mahasiswa)**  **print("\nData Mahasiswa yang Dimasukkan")**  **for mhs in mahasiswa\_list:**  **mhs.tampilkan\_info()** |

#### Source Code Soal 3

|  |
| --- |
| ***class* Kucing:**  **def \_\_init\_\_(self, nama, umur, warna):**  **self.nama = nama**  **self.umur = umur**  **self.warna = warna**  **def suara(self):**  **return f"{self.nama} mengeong: Meong!"**  **def info(self):**  **return f"Kucing bernama {self.nama}, umur {self.umur} tahun, warna {self.warna}."**  ***class* Anjing:**  **def \_\_init\_\_(self, nama, ras, umur):**  **self.nama = nama**  **self.ras = ras**  **self.umur = umur**  **def suara(self):**  **return f"{self.nama} menggonggong: Guk Guk!"**  **def info(self):**  **return f"Anjing bernama {self.nama}, ras {self.ras}, umur {self.umur} tahun."**  ***class* Burung:**  **def \_\_init\_\_(self, nama, jenis, warna\_bulu):**  **self.nama = nama**  **self.jenis = jenis**  **self.warna\_bulu = warna\_bulu**  **def suara(self):**  **return f"{self.nama} berkicau: Cuit cuit!"**  **def info(self):**  **return f"Burung bernama {self.nama}, jenis {self.jenis}, warna bulu {self.warna\_bulu}."**  **daftar\_hewan = []**  **for i in range(3):**  **print(f"\nInput data hewan ke-{i+1}:")**  **jenis = input("Pilih jenis hewan (kucing/anjing/burung): ")**  **if jenis == "kucing":**  **nama = input("Nama kucing: ")**  **umur = input("Umur kucing: ")**  **warna = input("Warna kucing: ")**  **kucing = Kucing(nama, umur, warna)**  **daftar\_hewan.append(kucing)**  **elif jenis == "anjing":**  **nama = input("Nama anjing: ")**  **ras = input("Ras anjing: ")**  **umur = input("Umur anjing: ")**  **anjing = Anjing(nama, ras, umur)**  **daftar\_hewan.append(anjing)**  **elif jenis == "burung":**  **nama = input("Nama burung: ")**  **jenis\_burung = input("Jenis burung: ")**  **warna = input("Warna bulu burung: ")**  **burung = Burung(nama, jenis\_burung, warna)**  **daftar\_hewan.append(burung)**  **else:**  **print("Jenis hewan tidak dikenali, lewati input ini.")**  **print("\n--- Informasi Semua Hewan ---")**  **for hewan in daftar\_hewan:**  **print(hewan.info())**  **print(hewan.suara())** |

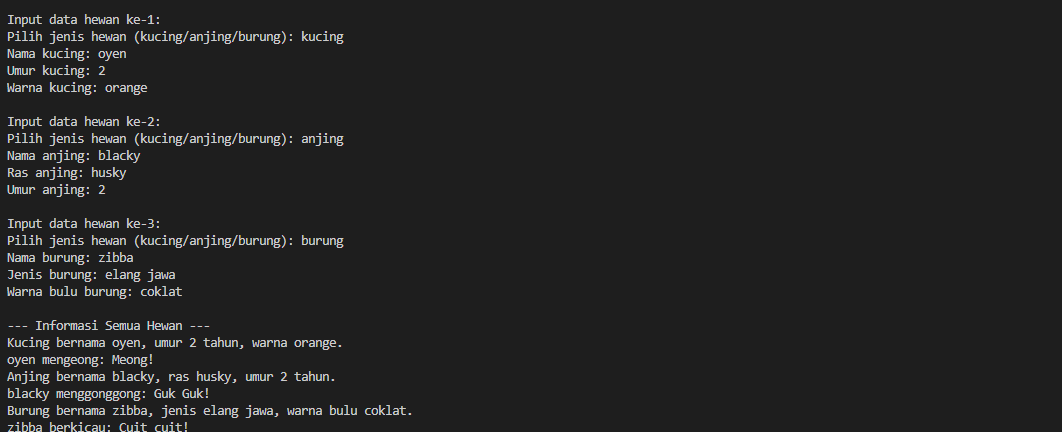
### Hasil

#### Hasil Soal 1



#### Hasil Soal 2

#### Hasil Soal 3

****

### Penjelasan

#### Penjelasan Soal 1

Pertama dibuat sebuah *class* bernama manusia yang merepresentasikan manusia secara umum. Di dalam *class* ini terdapat sebuah *constructor* \_\_init\_\_ yang digunakan untuk menginisialisasi atribut nama, umur, dan alamat ketika sebuah objek baru dibuat. Selain itu, ada dua metode yang mewakili perilaku manusia, yaitu berjalan() dan berlari(), yang akan mencetak aktivitas berdasarkan nama objek yang dipanggil.

Selanjutnya, kode membuat enam objek dari *class* manusia, masing-masing merepresentasikan individu berbeda dengan nama, umur, dan alamat yang sama (dalam contoh ini semua beralamat di Jakarta). Setelah objek dibuat, beberapa objek dipanggil metode berjalan(), sementara yang lainnya dipanggil metode berlari().

#### Penjelasan Soal 2

Kode program yang dibuat merupakan contoh praktis penggunaan konsep *class*, *object*, dan *constructor* dalam program yang berinteraksi langsung dengan input pengguna. Di sini, dibuat sebuah *class* bernama Mahasiswa yang merepresentasikan data mahasiswa. *Constructor* (\_\_init\_\_) digunakan untuk menginisialisasi atribut seperti nama, nim, jurusan, dan alamat saat objek dibuat. Selain itu, terdapat metode tampilkan\_info() yang bertugas untuk mencetak informasi lengkap dari masing-masing mahasiswa.

Program kemudian meminta pengguna untuk memasukkan data tiga mahasiswa melalui perulangan for. Setiap data yang dimasukkan langsung digunakan untuk membuat objek Mahasiswa, lalu objek tersebut disimpan ke dalam list mahasiswa\_list. Setelah semua data dikumpulkan, program menampilkan seluruh informasi mahasiswa menggunakan perulangan kedua yang memanggil metode tampilkan\_info() untuk setiap objek.

#### Penjelasan Soal 3

Program yang saya buat adalah contoh penerapan konsep OOP (*Object*-Oriented Programming) dalam Python, melibatkan beberapa *class* berbeda untuk merepresentasikan berbagai jenis hewan: Kucing, Anjing, dan Burung. Masing-masing *class* memiliki atribut unik yang sesuai dengan ciri khas hewan tersebut, serta metode info() untuk menampilkan deskripsi lengkap dan suara() untuk menampilkan suara khas dari hewan tersebut. Setiap *class* memiliki *constructor* (\_\_init\_\_) untuk menginisialisasi data saat objek dibuat.

Dalam bagian utama program, pengguna diminta untuk menginput data sebanyak tiga hewan. Melalui perulangan for, program akan menanyakan jenis hewan yang ingin dimasukkan, lalu meminta data spesifik sesuai jenisnya.

Jika pengguna memasukkan jenis yang dikenali (kucing, anjing, atau burung), maka akan dibuat objek dari *class* yang sesuai dan ditambahkan ke list daftar\_hewan. Setelah seluruh data dikumpulkan, program akan mencetak informasi lengkap dan suara dari setiap hewan dalam list tersebut dengan memanggil metode info() dan suara() secara dinamis.

## BAB V PENUTUP

### Analisa

Dari hasil praktikum, saya menganalisis bahwa *class* adalah kerangka atau cetakan yang digunakan untuk membuat objek dalam pemrograman berorientasi objek (OOP). *Class* mendefinisikan atribut dan *method* yang akan dimiliki oleh objek. Dengan menggunakan *class*, pengelolaan kode menjadi lebih terstruktur dan modular, sehingga memudahkan pengembangan dan pemeliharaan program.

Selain itu, *object* adalah instance dari sebuah *class* yang memiliki nilai atribut dan dapat menjalankan *method* yang telah didefinisikan dalam *class*. Untuk menginisialisasi objek, digunakan *constructor*, yaitu *method* khusus yang secara otomatis dipanggil saat objek dibuat. *Constructor* memungkinkan pemberian nilai awal pada atribut tanpa harus mengaturnya secara manual setelah objek dibuat.

### Kesimpulan

Dari hasil praktikum,saya menyimpulkan bahwa *class*, *object*, dan *constructor* adalah konsep fundamental dalam pemrograman berorientasi objek (OOP). *Class* berfungsi sebagai blueprint untuk membuat objek dengan atribut dan *method* tertentu. *Object* adalah instance dari *class* yang memiliki nilai atribut sendiri dan dapat menjalankan *method* sesuai dengan definisi *class*.

Selain itu, *constructor* mempermudah inisialisasi objek dengan mengatur nilai awal atribut saat objek dibuat. Dengan memahami konsep ini, pengembangan program menjadi lebih efisien dan terstruktur.

1. *Class* digunakan untuk mendefinisikan atribut dan *method* yang akan dimiliki oleh objek.
2. *Object* adalah hasil instansiasi dari *class* yang memiliki nilai atribut dan dapat menjalankan *method*.
3. *Constructor* secara otomatis menginisialisasi objek dengan nilai awal saat objek dibuat.

**LAPORAN RESMI**

# MODUL II *Methods (Instance, Class dan Static)*

**PEMROGRAMAN BERBASIS OBJEK**

**NAMA : ADYTTA PUTRA TARIGAN**

**N.R.P : 240441100139**

**DOSEN : YUDHA DWI PUTRA NEGARA, S.KOM., M.KOM.**

**ASISTEN : AHMAD RIKHAN ARBA’I**

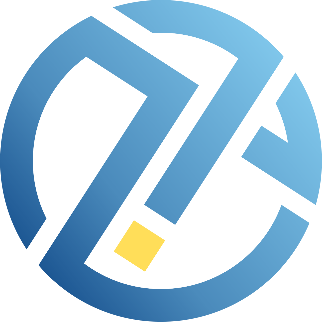
**TGL PRAKTIKUM : 12 APRIL 2025**

**Disetujui : 18 APRIL 2025**

**Asisten**

**AHMAD RIKHAN ARBA’I**

**23.04.411.00192**



**LABORATORIUM TEKNOLOGI INFORMASI**

**PRODI SISTEM INFORMASI**

**JURUSAN TEKNIK INFORMATIKA**

**FAKULTAS TEKNIK**

**UNIVERSITAS TRUNOJOYO MADURA**

## BAB I PENDAHULUAN

### Latar Belakang

Dalam pemrograman berorientasi objek, method adalah fungsi yang didefinisikan di dalam sebuah kelas dan digunakan untuk menetapkan perilaku objek. Di antara berbagai jenis method, tiga yang paling umum adalah instance method, class method, dan static method. Ketiganya memiliki perbedaan dalam cara akses dan penggunaannya terhadap data dalam kelas maupun objek, yang membuat masing-masing punya peran khusus dalam desain program.

Instance method merupakan method yang paling sering digunakan. Method ini hanya bisa dipanggil melalui objek hasil instansiasi dari suatu kelas. Dengan kata lain, instance method memiliki akses ke data dan atribut spesifik milik objek tersebut melalui parameter self. Sementara itu, class method memiliki akses ke kelas itu sendiri melalui parameter cls, dan digunakan untuk mengelola data yang bersifat umum atau milik bersama dari semua objek kelas tersebut.

Berbeda dari keduanya, static method tidak memiliki akses langsung ke instance (self) maupun kelas (cls). Static method biasanya digunakan untuk fungsi bantu yang berkaitan dengan kelas, namun tidak memerlukan informasi dari instance maupun kelas. Dengan memahami dan menerapkan ketiga jenis method ini secara tepat, pengembang dapat membangun struktur program yang rapi, modular, dan mudah dipelihara dalam jangka panjang.

### Tujuan

* Mahasiswa mampu memahami konsep Instance Method, Class Method dan Static Method dalam Pemrograman Berorientasi Objek serta mampu mengimplementasikannya.
* Mahasiswa mampu mengimplementasikan masing -vmasing jenis method
* Mahasiswa mampu mengidentifikasi jenis method yang sesuai berdasarkan kebutuhan fungsional dalam suatu kelas

## BAB II DASAR TEORI

### Methods (Instance, Class dan Static)

#### Methods

Method adalah fungsi yang didefinisikan dalam sebuah class dan dapat dipanggil oleh object dari kelas tersebut. Method memungkinkan objek untuk melakukan suatu aksi atau operasi dengan menggunakan data yang ada dalam objek itu sendiri.

Supaya mudah untuk dipahami lagi, bayangkan method adalah suatu perilaku dari objek, misalnya objeknya adalah mobil, maka mobil tersebut memiliki beberapa perilaku seperti, berjalan maju, berjalan mundur, menyalakan lampu sein dan sebagainya

Dalam Object Oriented Programming pada Python terdapat 3 jenis method yaitu: Instance Method, Class Method dan Static Method.

1. Ciri-Ciri Method pada Class

a. Method didefinisikan di dalam kelas dan memiliki fungsi untuk menggambarkan perilaku atau aksi yang dimiliki oleh objek atau kelas.

b. Method dapat dipanggil melalui kelas atau objek, tergantung pada jenis method yang digunakan.

c. Memiliki dekorator untuk menandai jenis method

d. Method dapat memiliki parameter maupun tidak

2. Syntax Penulisan Method pada Class

Dalam Python, method merupakan perilaku yang dapat diberikan kepada instance class (object). Method ini dapat dibuat ketika membuat sebuah class seperti contoh di bawah.

Dari kode diatas dapat dilihat bahwa cara mendeklarasikan sebuah method dengan menggunakan <def> <nama method> : <perintah> atau dapat dilihat pada kotak berwarna kuning pada gambar diatas. Sedangkan untuk memanggil method dapat dilakukan dengan cara <nama\_objek/kelas>. <nama\_method> () atau dapat dilihat kotak merah pada gambar diatas. Mendeklarasikan dan memanggil method memiliki cara yang berbeda-beda tergantung dengan jenis methodnya. Jenis method yang terdapat dalam python sendiri memiliki 3 jenis yaitu Instance Method, Class Method dan Static Method.

#### Instance Method

Instance method adalah method yang paling umum digunakan dalam Python. Method ini didefinisikan dalam kelas dan diakses melalui objek dari kelas tersebut. Instance method beroperasi pada instance objek dan dapat mengakses atribut dan method lainnya di dalam kelas tersebut.

1. Ciri-ciri Instance Method

a. Deklarasinya di dalam block class

b. Parameter pertamanya adalah self

c. Method diakses menggunakan notasi <object>. <method> ()

2. Syntax Penulisan pada Instance Method.

1. Instance method dapat ditulis dengan menambahkan parameter self pada method yang akan digunakan sebagai instance method seperti :

2. Panggil Instance method yang sudah dibuat dalam kelas mamalia dengan cara <objek>.<nama\_method>() seperti :

pada kelas tersebut memiliki beberapa method seperti mengatur nama, dan juga menampilkan informasi pada setiap intance (objek) kelas tersebut.

parameter self pada setiap method mengacu pada setiap objek yang akan dibuat dari kelas mamalia. hal ini lah yang menyebabkan setiap objek memiliki nilai attribute yang berbeda beda sehingga akan mengahsilkan output:

#### Class Method

Class method adalah method yang pemiliknya adalah class dengan pengaksesan adalah via class, berbeda dibanding instance method yang diperuntukan untuk instance object. Jika instance method memiliki parameter self yang isinya adalah instance object, maka class method memiliki parameter cls yang isinya adalah tipe

1. Ciri-Ciri Class Method

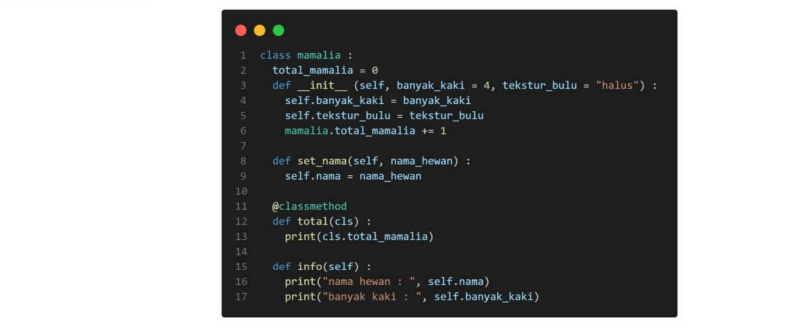
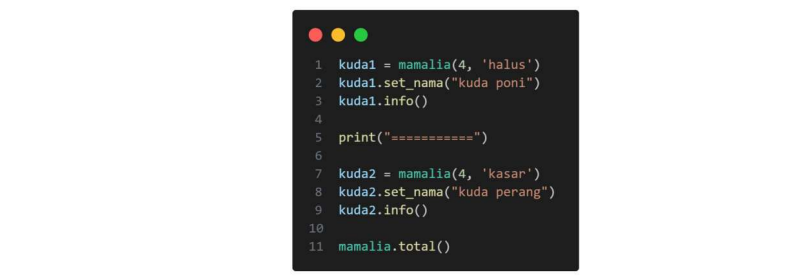
a. Class method memiliki parameter pertama bernama cls yang merupakan alias dari class dimana method tersebut dideklarasikan (misalnya class method dideklarasikan dalam class Car, maka paramter cls berisi tipe data class Car).

b. Selain perbedaan di atas, class method dideklarasikan dengan menuliskan decorator @classmethod

2. Syntax Penulisan Class Method

Class method dapat digunakan untuk beberapa tujuan seperti mengakses class variable ataupun membuat objek baru dengan menggunakan parameter cls. Berikut adalah beberapa cara penulisan code class method untuk mengakses class variable maupun membuat objek baru.

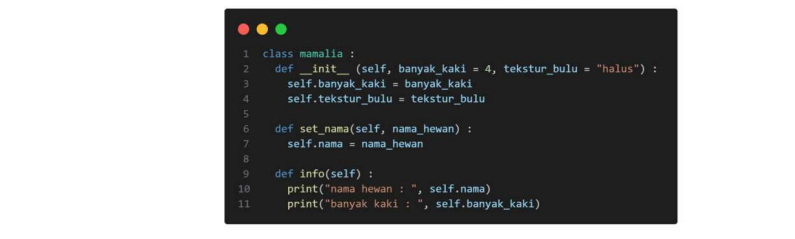
2a. Cara mengakses class variable

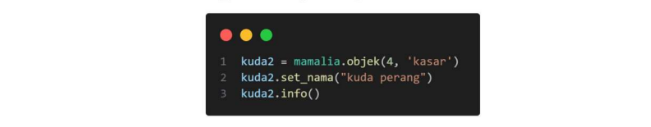
1. Buat class dan method seperti pada contoh di bawah :
2. Kemudian tambahkan class attribute, class method (dengan menambahkan decorator @classmethod untuk mendeklarasikan class method) dan juga operator aritmatika pada constructor.data class
3. Buat objek dari kelas mamalia dan panggil class method seperti baris code dibawah

Output:

Dari hasil output diatas dapat dilihat bahwa total hewan mamalia yang telah dibuat adalah 2 ekor, sesuai dengan banyaknya objek yang telah kita buat kudal dan kuda2. hal ini karena ketika memanggil nama kelas mamalia.total () maka method total() akan mengakses class attribute yaitu total\_mamalia.

2b. Cara Membuat Objek dengan menggunakan parameter CLS

1. Buat class dengan nama mamalia seperti pada gambar.
2. Tambahkan class method seperti pada gambar.
3. Di sini anda dapat membuat objek dengan menggunakan parameter cls dengan beberapa cara seperti membuat objek biasa, menggunakan <nama\_kelas>. <class\_method> (). Atau dengan objek dari kelas yang sama <objek>. <class\_meethod> ().
4. Pertama coba dengan membuat objek seperti pada umunya.
5. Kedua, coba buat objek dengan memanggil <nama\_kelas>.<class\_method>().

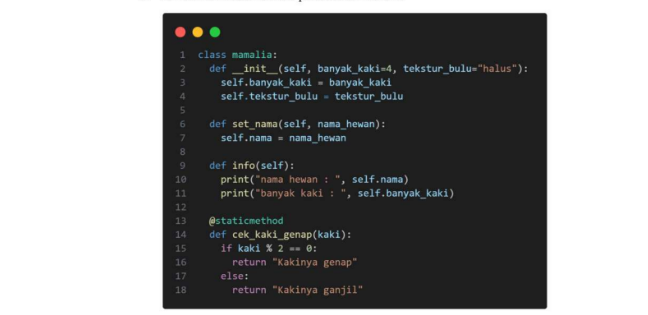


1. Terakhir, coba buat objek dengan menggunakan objek dari kelas yang sama.
2. Ketika di run akan menampilkan output :

Dengan menggunakan class method kita bisa membuat objek dengan berbagai cara. parameter cls mengacu pada kelas itu sendiri atau memanggil dirinya sendiri sehingga dapat membuat objek dengan menggunakan class method.

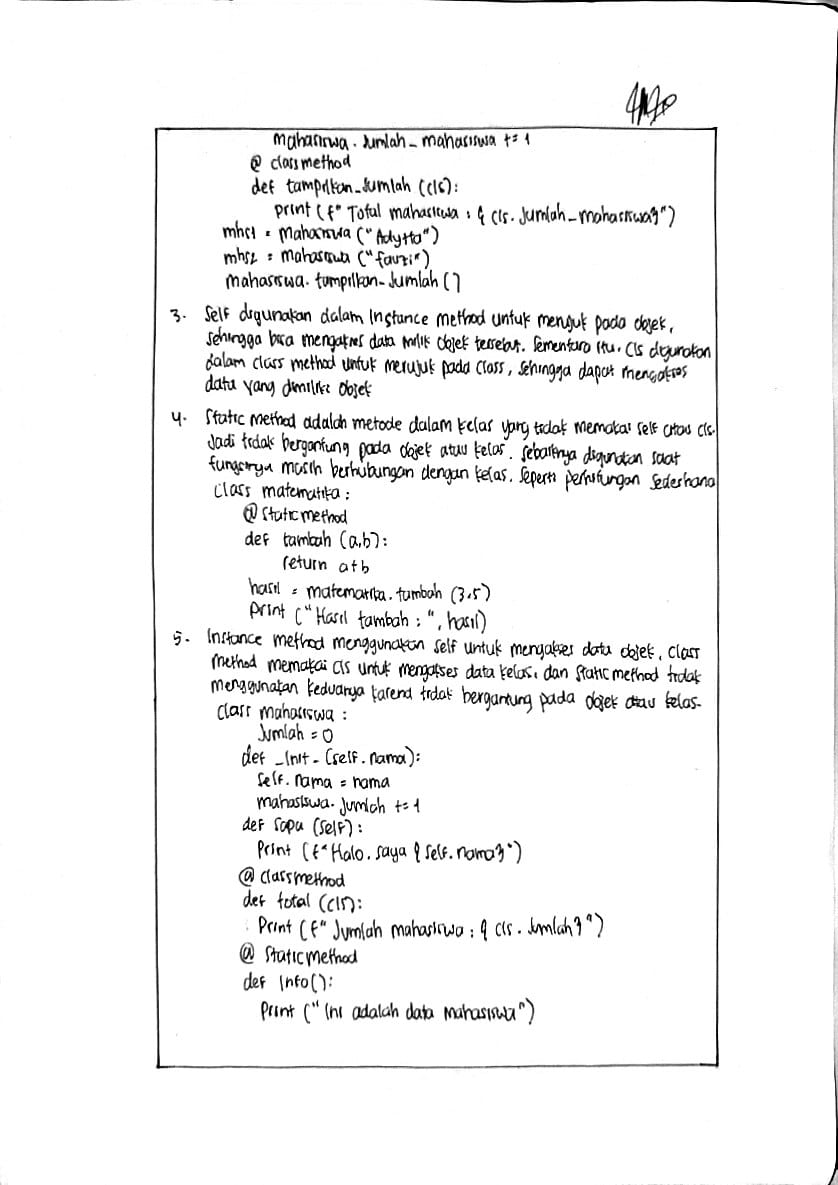
#### Static Method

Static method adalah metode dalam kelas yang tidak bergantung pada instance atau atribut kelas. Static method bekerja seperti fungsi biasa tetapi berada dalam ruang lingkup kelas. Metode ini tidak memiliki parameter self (untuk instance method) atau cls (untuk class method).

1. Ciri-ciri static method
2. Tidak bisa mengakses atau mengubah atribut instance (self).
3. Tidak bisa mengakses atau mengubah atribut kelas (cls).
4. Bisa dipanggil langsung dari kelas tanpa membuat instance.
5. Biasanya digunakan untuk fungsi yang berhubungan dengan kelas tetapi tidak memerlukan data dari instance atau kelas itu sendiri.
6. Menggunakan decorator @staticmethod.
7. Syntax Penulisan Static Method
8. Buat class dengan nama mamalia seperti pada gambar.
9. Tambahkan static method pada class mamalia.
10. Buat objek dari class mamalia.
11. Panggil static method dengan menggunakan <nama\_kelas>.<static\_method>().

Output :

## BAB III TUGAS PENDAHULUAN



## BAB IV IMPLEMENTASI

### Tugas Praktikum

#### Tugas Praktikum No. 1

1. Mahasiswa

Digunakan untuk

1. Menyimpan nama, NIM, prodi, dan daftar mata kuliah yang diambil
2. Bisa menambahkan mata kuliah ke daftar
3. Bisa menampilkan biodata + matkul yang diambil
4. Bisa menampilkan jumlah mahasiswa yang telah dibuat(class method).
5. Bisa memvalidasi apakah NIM-nya benar atau tidak (static method). – NIM harus dimulai dengan “23” dan total 10 digit.
6. MataKuliah

Atribut:

1. Kode mata kuliah, nama, dan jumlah SKS.

Fitur tambahan:

1. Bisa mengecek apakah SKS valid atau tidak (static method).

* SKS yang boleh hanya 2 atau 3.

1. Kampus

Menyimpan data kampus:

1. Atributnya: nama kampus dan alamat kampus.
2. Ada atribut class yang menyimpan total mahasiswa (jumlah\_mahasiswa)
3. Bisa menampilkan nama kampus dan total mahasiswa (class method).
4. Bisa mengecek apakah nama kampus valid (tidak mengandung angka) (static method).

NB:

1. Wajib buat:
2. Minimal 6 objek mahasiswa.
3. Minimal 8 objek matakuliah
4. Minimal 1 objek kampus
5. Setiap mahasiswa harus mengambil minimal 4 mata kuliah.
6. Validasi:
7. NIM dicek saat buat objek mahasiswa
8. SKS dicek saat buat objek matakuliah.
9. Nama kampus dicek saat buat objek kampus.
10. Terakhir, tampilkan semua:
11. Info mahasiswa + matkul yang mereka ambil.
12. Info kampus + cek apakah nama kampus valid

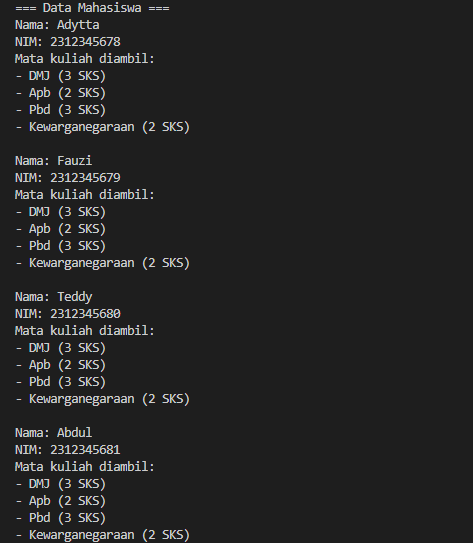
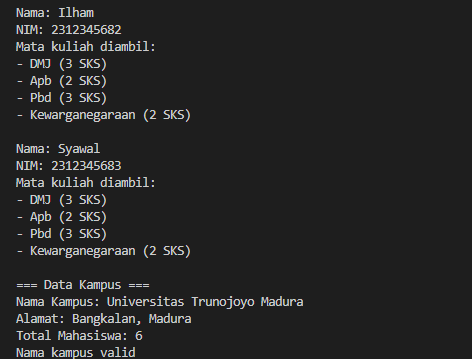
### Source Code

#### Source Code Soal 1

|  |
| --- |
| class Mahasiswa:  jumlah\_mahasiswa = 0  def \_\_init\_\_(self, nama, nim):  if Mahasiswa.validasi\_nim(nim):  self.nama = nama  self.nim = nim  Mahasiswa.jumlah\_mahasiswa += 1  Kampus.jumlah\_mahasiswa += 1  else:  self.nama = "Tidak Valid"  self.nim = "-"  self.matkul = []  def tambah\_matkul(self, mk):  self.matkul.append(mk)  def info(self):  print("Nama:", self.nama)  print("NIM:", self.nim)  print("Mata kuliah diambil:")  for m in self.matkul:  print("-", m.nama, f"({m.sks} SKS)")  print()  @classmethod  def total\_mahasiswa(cls):  return cls.jumlah\_mahasiswa  @staticmethod  def validasi\_nim(nim):  return nim.startswith("23") and len(nim) == 10 and nim.isdigit()  class MataKuliah:  def \_\_init\_\_(self, nama, sks):  if MataKuliah.cek\_sks\_valid(sks):  self.nama = nama  self.sks = sks  else:  self.nama = "Tidak Valid"  self.sks = 0  print("SKS tidak valid")  @staticmethod  def cek\_sks\_valid(sks):  return sks in [2, 3]  class Kampus:  jumlah\_mahasiswa = 0  def \_\_init\_\_(self, nama, alamat):  self.nama = nama  self.alamat = alamat  def info(self):  print("Nama Kampus:", self.nama)  print("Alamat:", self.alamat)  print("Total Mahasiswa:", Kampus.jumlah\_mahasiswa)  @classmethod  def info\_kampus(cls, nama):  print(f"Nama Kampus: {nama}")  print(f"Total Mahasiswa: {cls.jumlah\_mahasiswa}")  @staticmethod  def cek\_nama\_valid\_static(nama):  if any(char.isdigit() for char in nama):  print("Nama kampus tidak valid (mengandung angka)")  else:  print("Nama kampus valid")  kampus = Kampus("Universitas Trunojoyo Madura", "Bangkalan, Madura")  matkuls = [  MataKuliah("DMJ", 3),  MataKuliah("Apb", 2),  MataKuliah("Pbd", 3),  MataKuliah("Kewarganegaraan", 2),  MataKuliah("Inggris", 2),  MataKuliah("Pemrograman", 3),  MataKuliah("Basis Data", 3),  MataKuliah("Algoritma", 3)  ]  m1 = Mahasiswa("Adytta", "2312345678")  m2 = Mahasiswa("Fauzi", "2312345679")  m3 = Mahasiswa("Teddy", "2312345680")  m4 = Mahasiswa("Abdul", "2312345681")  m5 = Mahasiswa("Ilham", "2312345682")  m6 = Mahasiswa("Syawal", "2312345683")  semua\_mhs = [m1, m2, m3, m4, m5, m6]  for mhs in semua\_mhs:  mhs.tambah\_matkul(matkuls[0])  mhs.tambah\_matkul(matkuls[1])  mhs.tambah\_matkul(matkuls[2])  mhs.tambah\_matkul(matkuls[3])  print("=== Data Mahasiswa ===")  for mhs in semua\_mhs:  mhs.info()  print("=== Data Kampus ===")  kampus.info()  Kampus.cek\_nama\_valid\_static(kampus.nama) |

### Hasil

#### Hasil Soal 1



### Penjelasan

#### Penjelasan Soal 1

Soal kali ini saya membuat program yang mencakup tiga kelas utama: Kampus, Mahasiswa, dan MataKuliah, yang masing-masing berfungsi untuk mengelola data yang berbeda dan saling terhubung. Class Kampus bertugas untuk menyimpan informasi dasar tentang kampus, seperti nama dan alamat kampus, serta mengelola jumlah mahasiswa yang terdaftar di kampus tersebut. Ada validasi untuk memastikan bahwa nama kampus tidak mengandung angka, yang bertujuan menjaga konsistensi dan memastikan nama kampus sesuai dengan standar yang berlaku.

Class MataKuliah menyimpan informasi mengenai mata kuliah yang ditawarkan oleh kampus, termasuk nama mata kuliah dan jumlah SKS (Satuan Kredit Semester) yang dimiliki mata kuliah tersebut. Terdapat validasi yang memeriksa bahwa jumlah SKS yang dimasukkan bernilai positif; jika SKS yang dimasukkan bernilai nol atau negatif, mata kuliah dianggap tidak valid dan tidak akan disimpan.

Class Mahasiswa menyimpan informasi pribadi mahasiswa, seperti nama, NIM (Nomor Induk Mahasiswa), dan daftar mata kuliah yang mereka ambil. NIM mahasiswa juga divalidasi untuk memastikan bahwa NIM terdiri dari minimal 5 digit dan hanya berisi angka. Jika NIM tidak valid, mahasiswa tersebut tidak akan terdaftar dalam sistem. Namun, jika NIM valid, mahasiswa akan terdaftar secara otomatis di dalam class Kampus, yang mengupdate jumlah mahasiswa yang terdaftar di kampus tersebut.

Setelah data mahasiswa dan mata kuliah dimasukkan, program akan menampilkan informasi lengkap setiap mahasiswa beserta mata kuliah yang mereka ambil. Di akhir program, informasi mengenai kampus, termasuk nama, alamat, dan jumlah mahasiswa yang terdaftar, akan ditampilkan, bersama dengan validasi nama kampus untuk memastikan bahwa data yang ada sesuai dengan ketentuan yang telah ditentukan.

## BAB V PENUTUP

### Analisa

Dalam OOP Python, method dibagi menjadi tiga jenis utama: instance, class, dan static method. Ketiganya memiliki perbedaan dalam cara kerja dan tujuan penggunaannya. Instance method adalah yang paling umum, digunakan untuk mengakses dan mengelola data milik objek (instansinya) menggunakan parameter self. Sementara itu, class method menggunakan parameter cls dan beroperasi pada level class, bukan objek, sehingga cocok untuk mengelola data atau perilaku yang berlaku untuk semua objek class tersebut.

Static method, di sisi lain, tidak memerlukan akses ke data objek maupun class. Dengan menggunakan dekorator @staticmethod, method ini digunakan untuk menjalankan fungsi yang logikanya masih berkaitan dengan class tapi tidak tergantung pada data internalnya.

### Kesimpulan

Dari modul ini,saya belajar kalau instance method cocok untuk manipulasi data objek, class method untuk manipulasi data class, dan static method untuk fungsi umum yang bersifat netral. Pembedaan ini membantu dalam mengorganisir fungsi-fungsi di dalam class agar lebih jelas tujuannya.

Memahami ketiga jenis method ini adalah fondasi penting dalam membangun program berbasis OOP yang kuat. Dengan menempatkan method di tempat yang tepat, kita bisa membuat kode yang lebih bersih, terarah, dan mudah dipelihara ke depannya. Ini juga jadi bekal bagus buat ngoding di level yang lebih kompleks nanti.

1. Instance method digunakan untuk mengakses atau memodifikasi data milik objek, dan selalu menggunakan parameter self
2. Class method bekerja di level class, menggunakan @classmethod dan parameter cls untuk mengakses atau mengubah data class (bukan objek).
3. Static method tidak bergantung pada objek atau class, menggunakan @staticmethod, dan cocok untuk fungsi-fungsi logika

**LAPORAN RESMI**

# MODUL III INHERITANCE & OVERRIDING

**PEMROGRAMAN BERBASIS OBJEK**

**NAMA : ADYTTA PUTRA TARIGAN**

**N.R.P : 240441100139**

**DOSEN : YUDHA DWI PUTRA NEGARA, S.KOM., M.KOM.**

**ASISTEN : AHMAD RIKHAN ARBA’I**

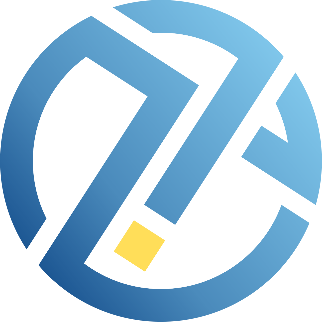
**TGL PRAKTIKUM : 26 APRIL 2025**

**Disetujui : 1 MEI 2025**

**Asisten**

**AHMAD RIKHAN ARBA’I**

**23.04.411.00192**



**LABORATORIUM TEKNOLOGI INFORMASI**

**PRODI SISTEM INFORMASI**

**JURUSAN TEKNIK INFORMATIKA**

**FAKULTAS TEKNIK**

**UNIVERSITAS TRUNOJOYO MADURA**

## **BAB I PENDAHULUAN**

### Latar Belakang

Inheritance atau pewarisan dalam pemrograman berorientasi objek adalah konsep di mana sebuah kelas (anak) bisa mewarisi properti dan metode dari kelas lain (induk). Tujuannya adalah untuk menghindari penulisan ulang kode yang sama, sehingga kode menjadi lebih efisien dan terstruktur. Misalnya, kalau kita punya kelas Hewan yang punya metode bergerak(), maka kelas Kucing atau Burung bisa mewarisi metode tersebut tanpa perlu menulis ulang, karena pada dasarnya mereka juga termasuk hewan yang bisa bergerak.

Namun dalam praktiknya, tidak semua metode dari kelas induk selalu cocok 100% untuk kelas anak. Di sinilah konsep Overriding menjadi penting. Overriding berarti kelas anak menulis ulang (menimpa) metode yang diwarisi dari kelas induk supaya lebih sesuai dengan kebutuhan spesifik. Misalnya, metode bergerak() di kelas Burung bisa ditimpa agar mencerminkan bahwa burung bergerak dengan cara terbang, bukan berjalan seperti kucing. Ini memberi fleksibilitas dan kendali lebih besar terhadap perilaku masing-masing objek.

Dengan kombinasi inheritance dan overriding, kita bisa membuat sistem program yang rapi, modular, dan mudah diperluas. Bayangkan jika setiap kelas harus ditulis dari nol tanpa bisa mewarisi apa pun—hal itu akan membuat program cepat menjadi rumit dan susah dipelihara. Inheritance menghemat waktu dan memperjelas hubungan antar objek, sedangkan overriding memberikan keleluasaan dalam menyesuaikan perilaku sesuai konteks masing-masing. Keduanya adalah fondasi penting dalam membuat program yang cerdas dan adaptif.

### Tujuan

* Mahasiswa mampu memahami konsep Inheritance dan Overriding dalam Pemrograman mengimplementasikannya. Berorientasi Objek serta mampu
* Mahasiswa mampu memahami Mengidentifikasi struktur hubungan antara superclass (parent class) dan subclass (child class).

## BAB II DASAR TEORI

### Pengertian Inheritance

Inheritance adalah salah satu konsep fundamental dalam Pemrograman berorientasi Objek. Konsep ini memungkinkan sebuah kelas (class) untuk mewarisi properti (atribut) dan perilaku (metode) dari kelas lain. Dengan kata lain, inheritance memungkinkan kita membuat kelas baru yang merupakan turunan dari kelas yang sudah ada, sehingga kelas baru tersebut dapat menggunakan kembali kode yang sudah ada tanpa harus menulis ulang.

#### Mengapa Inheritance Penting

1. Menghemat Waktu dan Tenaga (Reusabilitas Kode)

Dengan inheritance, kita bisa menggunakan kembali kode yang sudah dibuat di kelas sebelumnya. Jadi, kita tidak perlu menulis ulang kode yang sama berulang-ulang. Ini membuat pekerjaan kita jadi lebih cepat dan efisien.

1. Membuat Kode Lebih Rapi dan Teratur (Organisasi Kode)

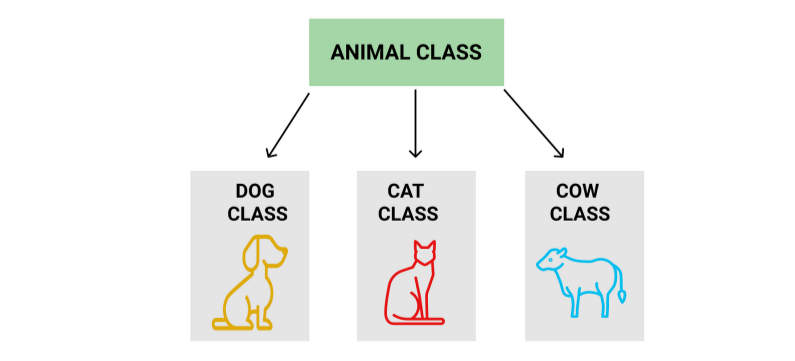
Dengan inheritance, kita bisa mengelompokkan kode berdasarkan hubungan yang mirip seperti di dunia nyata. Misalnya, semua hewan punya sifat dasar yang sama, jadi kita buat satu kelas hewan, lalu hewan hewan spesifik seperti anjing atau kucing dibuat sebagai turunan dari kelas hewan. Ini membuat kode kita lebih mudah dimengerti dan diatur.

1. Mempermudah Perubahan dan Perbaikan (Pemeliharaan Kode)

Kalau kita ingin mengubah sesuatu yang umum, kita cukup ubah di kelas induk saja. Perubahan ini otomatis akan berlaku ke semua kelas turunan yang mewarisinya. Jadi, kita tidak perlu mengubah kode di banyak tempat sekaligus.

1. Membuat Program Lebih Fleksibel (Polimorfisme)

Dengan inheritance, kita bisa menggunakan objek dari kelas turunan seolah-olah itu objek dari kelas induk. Ini membantu kita membuat program yang lebih fleksibel dan mudah dikembangkan, karena bisa menggunakan berbagai jenis objek yang berbeda tapi dengan cara yang sama.

****

Penjelasan :

1. ***Animal Class***: Merupakan kelas induk (*parent class*).yang dapat berisikan method yang akan diwariskan.
2. ***Dog Class, Cat Class, Cow Class*:** Merupakan kelas anak (*child* *class*) yang mewarisi atribut dan *method* dari *Animal* *Class*.

Contoh Kode:

1. **Parent Class**

****

*Class Animal* merupakan *parent class* (kelas induk) yang berfungsi sebagai dasar bagi kelas-kelas lain yang akan dibuat sebagai *child class* (kelas turunan). *Class* ini memiliki dua *method* (fungsi) yaitu Makan dan Jalan. yang mendefinisikan perilaku umum yang dapat dilakukan oleh semua objek hewan.

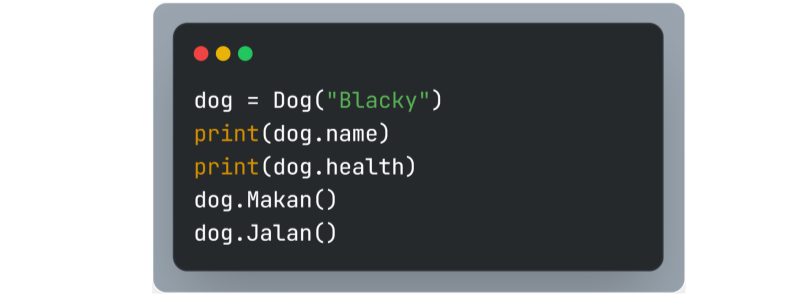
1. ***Child Class***

******

*Class Dog* merupakan *child class* atau kelas anak dari *class Animal*.

Artinya, *Dog* mewarisi semua atribut dan *method* yang sudah didefinisikan di *class* *Animal*.

* **Cara Memanggil**

****

* **Hasil**

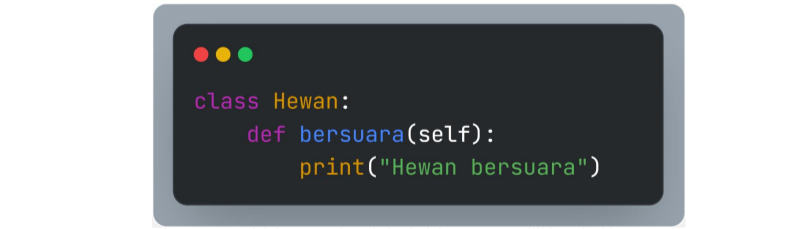


### Pengertian Overriding

*Overriding* adalah proses di mana sebuah child class (kelas anak) menulis ulang atau mengganti implementasi sebuah metode yang diwarisi dari parent class (kelas induk). Tujuannya adalah agar *child class* dapat memiliki perilaku atau fungsi yang berbeda dan lebih spesifik sesuai dengan kebutuhan atau karakteristiknya sendiri, meskipun metode tersebut sudah ada di kelas induk.

Contoh Kode:

1. ***Parent Class***



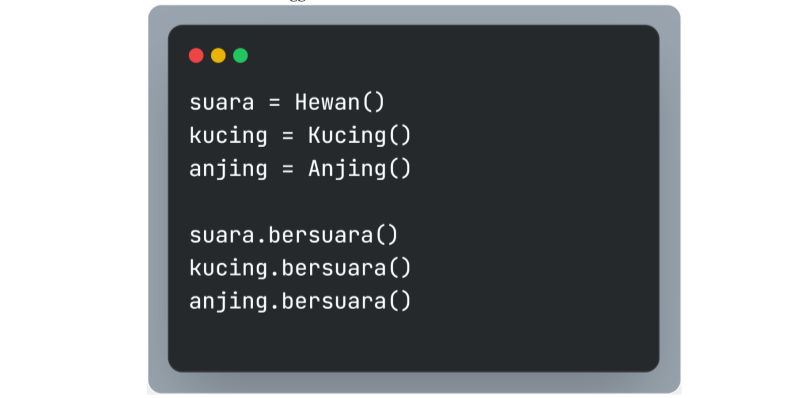
*Class* Hewan adalah *parent class* (kelas induk ) yang memiliki *method* bersuara.

* ***Overriding***



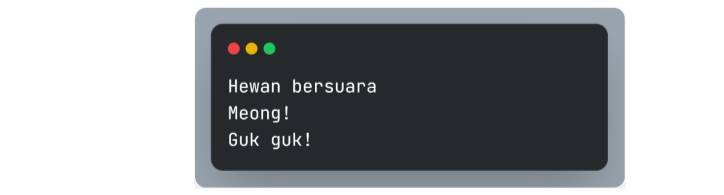
*Class* Kucing dan anjing merupakan *child class* dari kelas Hewan. Di sini, kita menulis ulang (*override*) *method* bersuara() yang ada di kelas Hewan. Jadi, meskipun Hewan punya *method* bersuara() yang menampilkan "Hewan bersuara", kelas Kucing menggantinya dengan *method* baru yang menampilkan "Meong!".Ini artinya, saat kita memanggil bersuara() pada objek Kucing, yang keluar bukan lagi "Hewan bersuara", tapi "Meong!". ini berlaku juga di Class Anjing.

1. **Cara Memanggil**

****

Untuk pemanggilannya kita membuat objek yang memanggil nama *class* setelah membuat objek kita memanggil *method*nya dengan memanggil objek titik lalu nama *method* yang akan dijalankan.

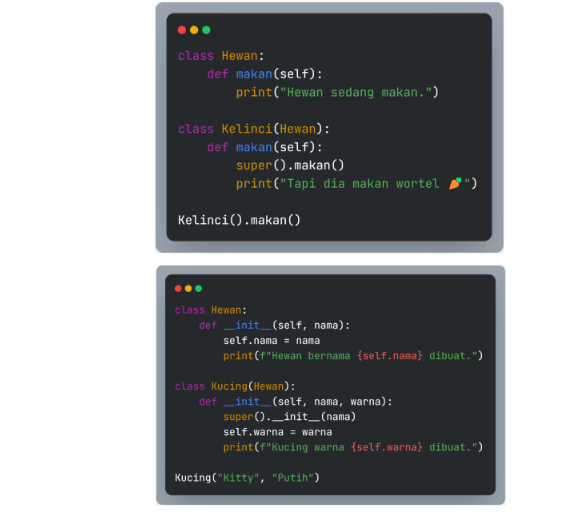
1. **Hasil**

****

Nah, yang awalnya *method* suara yang berisikan “Hewan Bersuara” setelah di *overriding* hasilnya berubah menjadi “Meong!” dan “Guk guk!”.

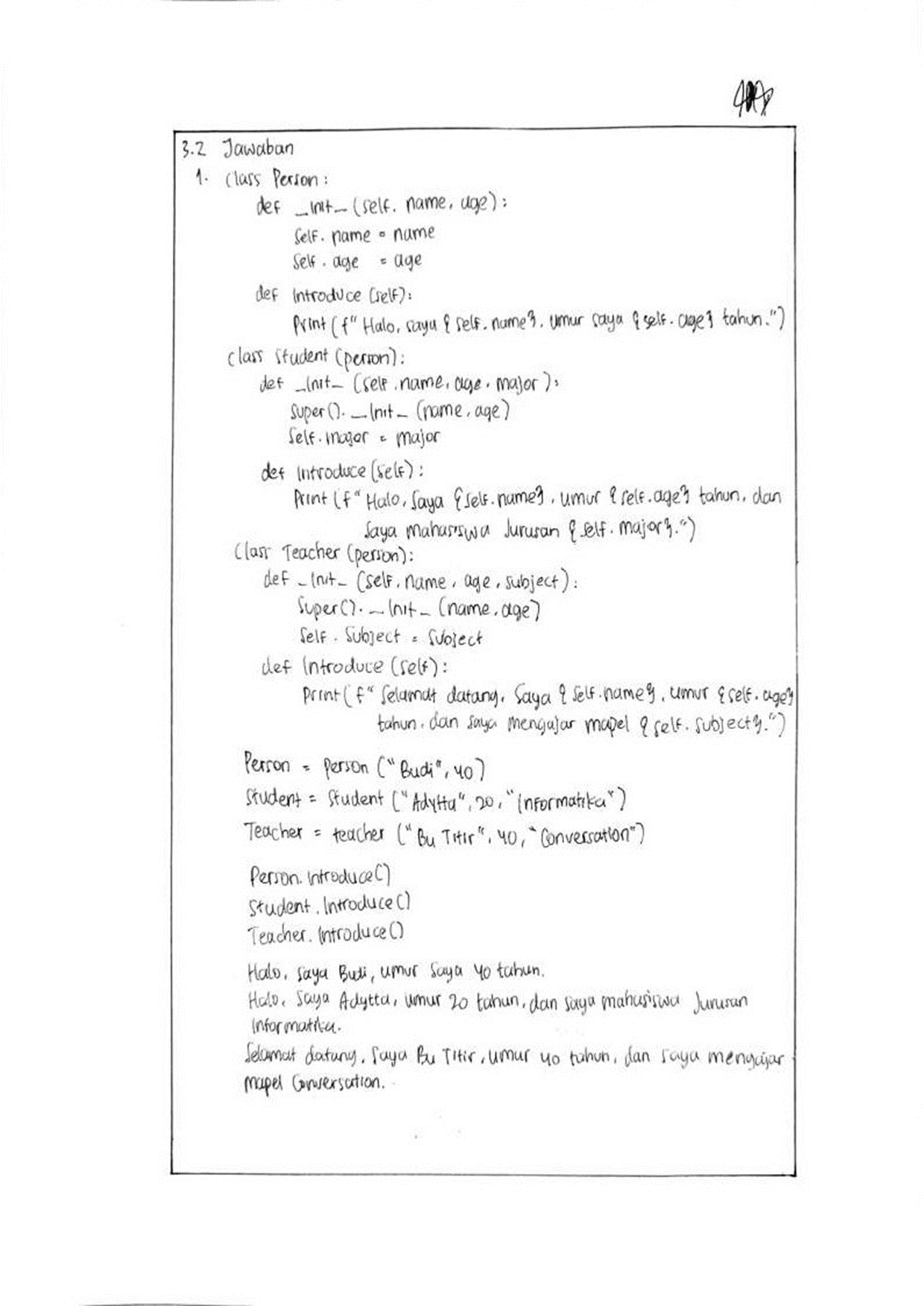
### Pengertian Super()

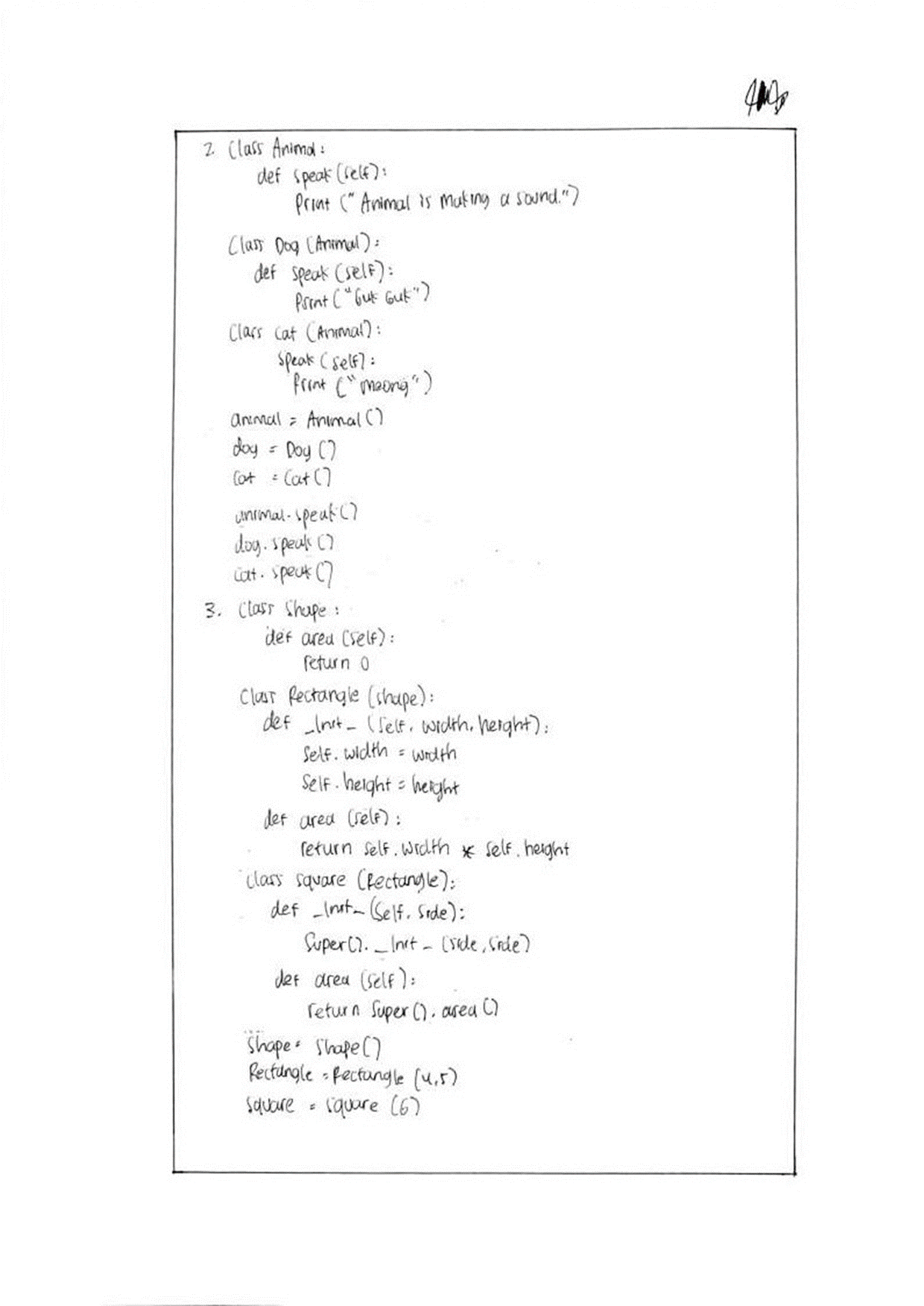
*super*() adalah fungsi bawaan yang digunakan untuk mengakses *method constructor* dari kelas induk (*parent class*) secara langsung, tanpa menyebutkan nama kelas induk tersebut secara *eksplisit*. Fungsi ini biasanya digunakan di dalam kelas turunan untuk melanjutkan atau memperluas perilaku yang telah didefinisikan oleh kelas induk. Salah satu penggunaan utama *super*() adalah untuk memanggil *constructor* induk (\_\_init\_\_) dari dalam *constructor* anak, agar atribut atau logika inisialisasi yang ditetapkan oleh kelas induk tetap dijalankan.

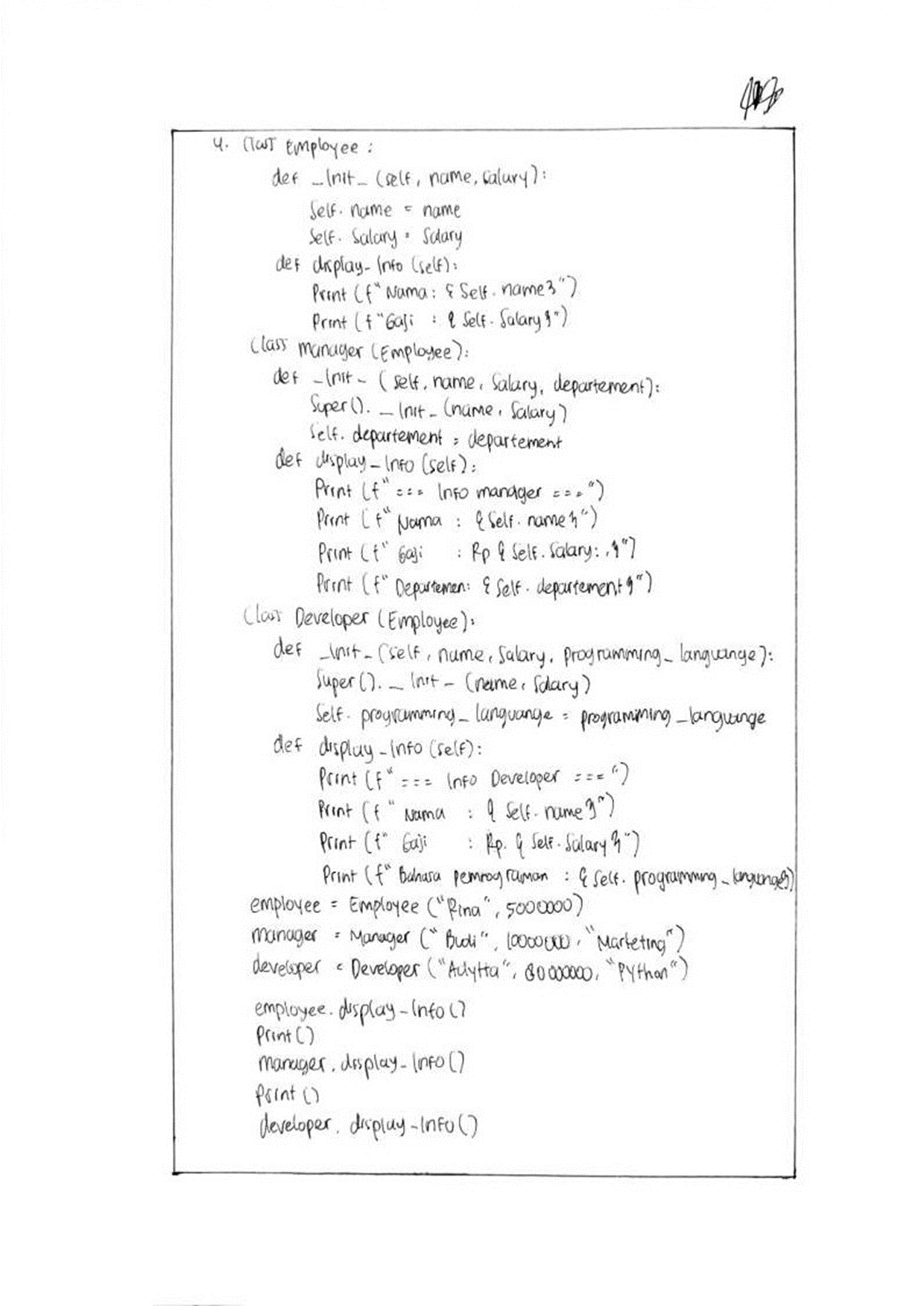
****

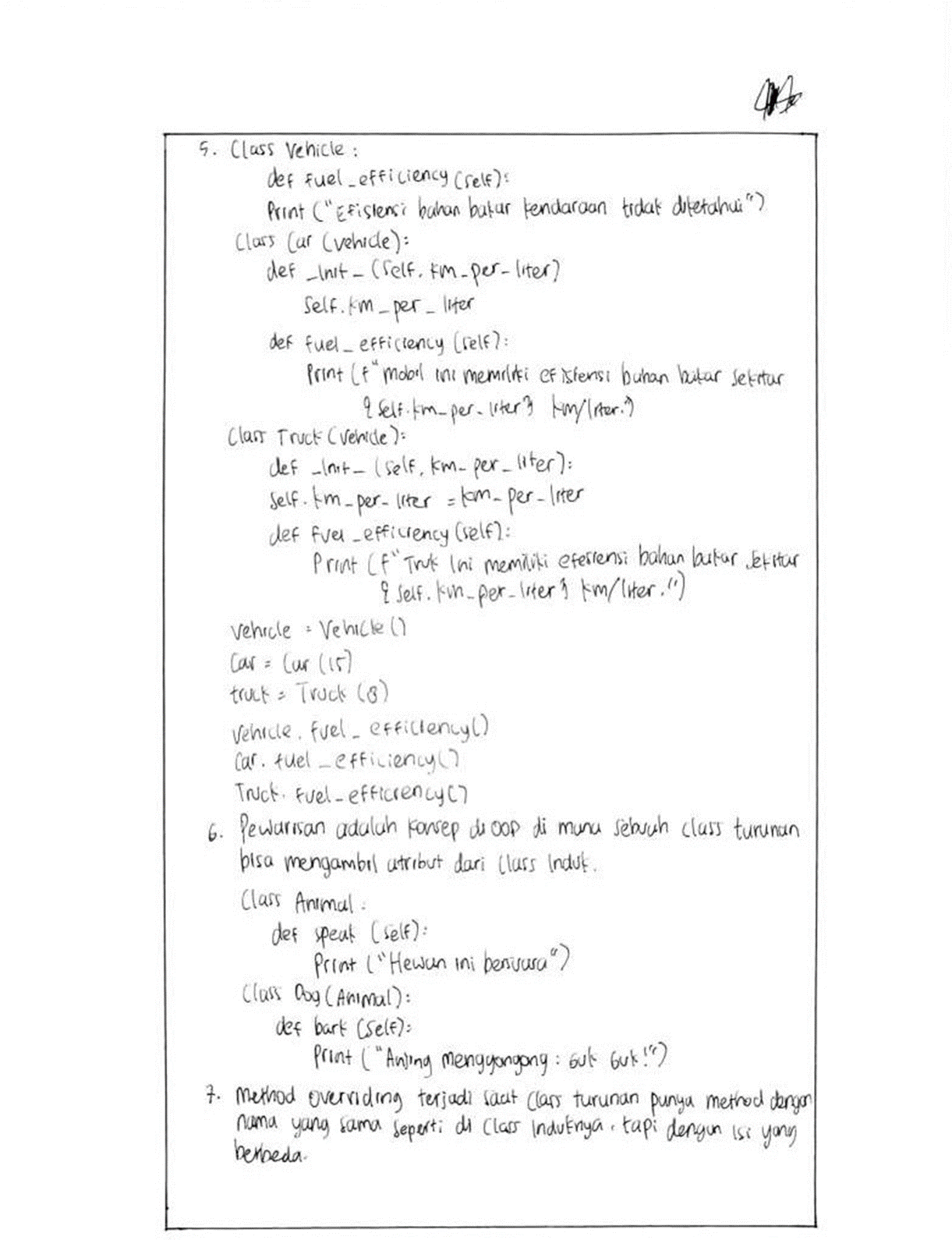
## BAB III TUGAS PENDAHULUAN

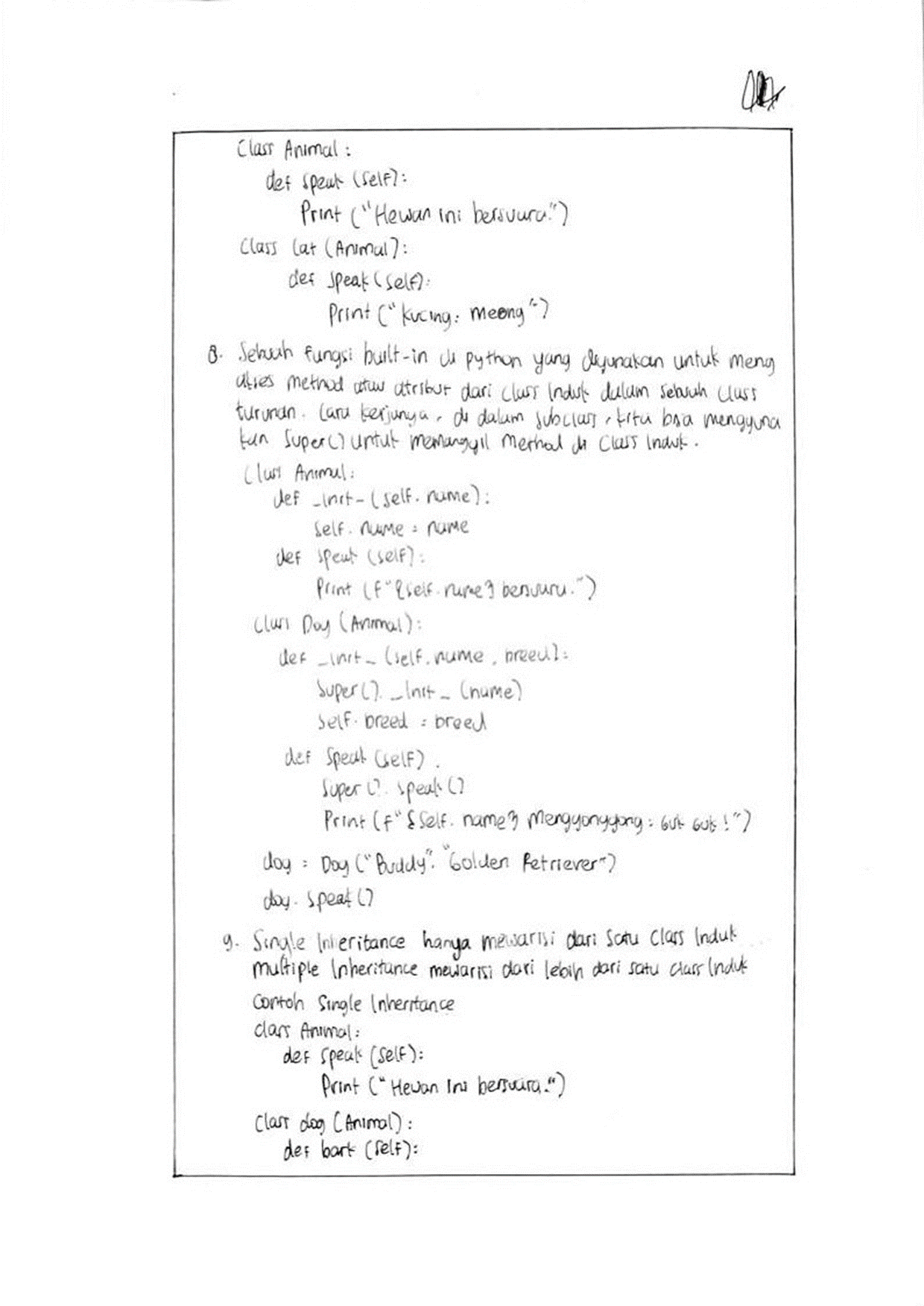
kami dalam memahami materi,

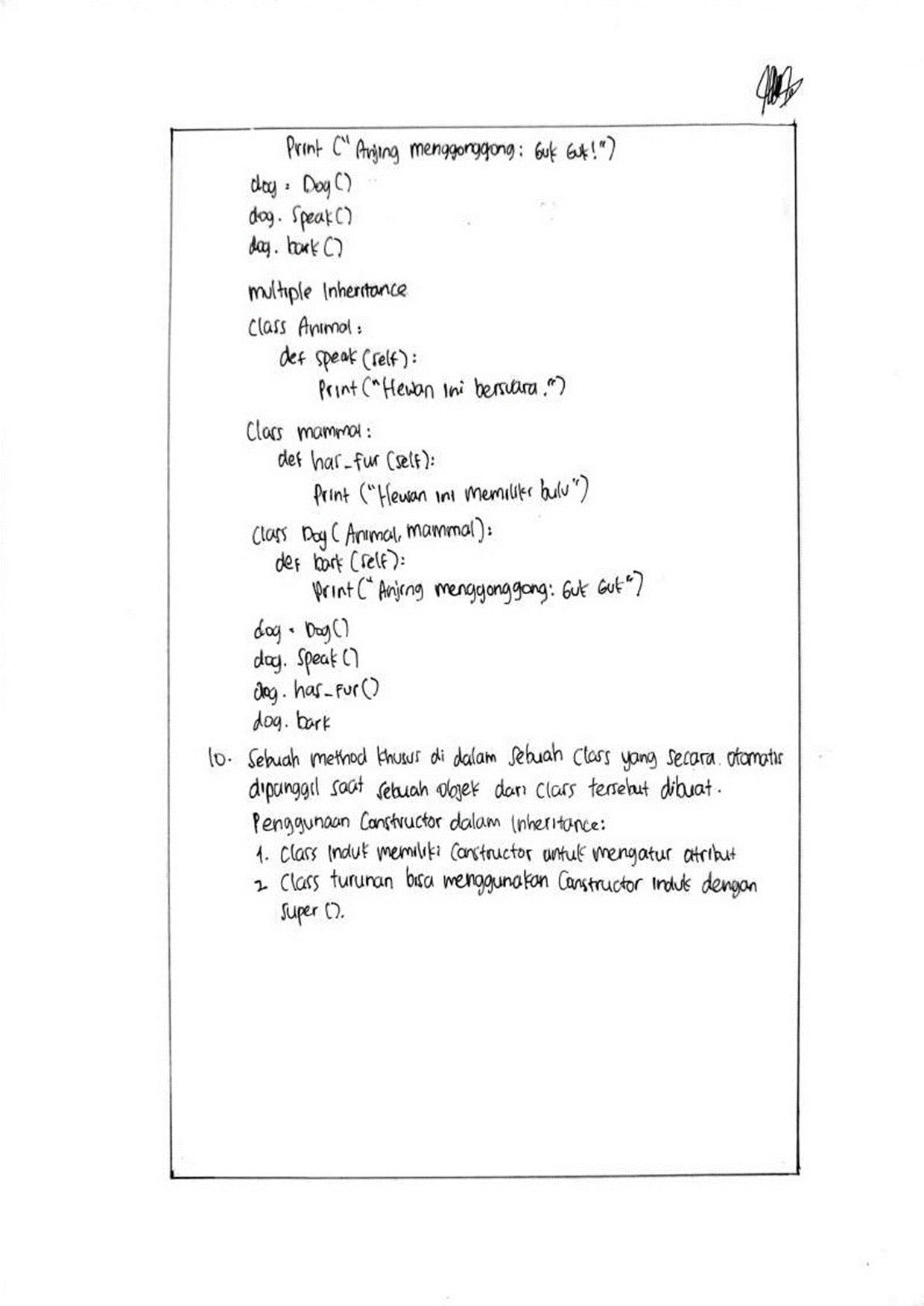
****

****

****

****

****

****

## BAB IV IMPLEMENTASI

### Tugas Praktikum

#### 4.1.1 Tugas Praktikum No. 1

Buatlah class Karyawan dengan atribut :

* nama, gaji, dan departemen
* serta method info() yang mencetak nama, gaji, dan departemen.

Buatlah class KaryawanTetap yang mewarisi Karyawan dengan atribut tambahan tunjangan dan *method* info() yang mencetak informasi karyawan tetap dengan tunjangan.

Buat class KaryawanHarian yang mewarisi Karyawan dengan atribut tambahan jam\_kerja (misalnya: jam kerja per hari) dan *method* info() yang mencetak informasi karyawan harian.

Buat class ManajemenKaryawan yang memiliki atribut :

* daftar\_karyawan, yang berupa list yang menyimpan objek KaryawanTetap dan KaryawanHarian.
* Tambahkan method tambah\_karyawan() di class ManajemenKaryawan untuk menambahkan objek karyawan ke dalam list daftar\_karyawan.
* Tambahkan method tampilkan\_semua\_karyawan() yang akan menampilkan informasi tentang semua karyawan dengan memanggil *method* info() untuk masing-masing objek karyawan di dalam list daftar\_karyawan.

Buatlah objek dari class ManajemenKaryawan dan tambahkan beberapa karyawan tetap dan harian.

Tampilkan informasi semua karyawan yang ada dalam daftar dengan memanggil method tampilkan\_semua\_karyawan().

#### Tugas Praktikum No. 2

Terapkan

Buatlah class Pengiriman dengan:

* Atribut asal dan tujuan.
* Method estimasi\_waktu() yang mengembalikan waktu estimasi pengiriman (misal: 5 hari).

Buatlah class PengirimanDarat yang mewarisi Pengiriman, dengan:

* Atribut tambahan jenis\_kendaraan.
* Override method estimasi\_waktu() berdasarkan jenis kendaraan Buatlah class PengirimanUdara yang mewarisi Pengiriman, dengan:
* Atribut tambahan maskapai.
* Override method estimasi\_waktu() berdasarkan maskapai.

Buatlah class PengirimanInternasional yang mewarisi PengirimanDarat dan PengirimanUdara, dan override estimasi\_waktu() untuk menyesuaikan waktu pengiriman internasional berdasarkan asal dan tujuan (misal: jika tujuan luar negeri, tambahkan 3 hari dari estimasi sebelumnya).

Buatlah beberapa objek PengirimanInternasional, set atribut-atributnya, dan panggil method estimasi\_waktu() untuk menghitung waktu pengiriman.

### Source code

#### Source Code Soal 1

|  |
| --- |
| class Karyawan:  def \_\_init\_\_(self, nama, gaji, departemen):  self.nama = nama  self.gaji = gaji  self.departemen = departemen  def tampilkan\_info(self):  return f"Nama: {self.nama}, Gaji: {self.gaji}, Departemen: {self.departemen}"  class KaryawanTetap(Karyawan):  def \_\_init\_\_(self, nama, gaji, departemen, tunjangan):  super().\_\_init\_\_(nama, gaji, departemen)  self.tunjangan = tunjangan  def tampilkan\_info(self):  return f"{super().tampilkan\_info()}, Tunjangan: {self.tunjangan}"  class KaryawanHarian(Karyawan):  def \_\_init\_\_(self, nama, gaji, departemen, jam\_kerja):  super().\_\_init\_\_(nama, gaji, departemen)  self.jam\_kerja = jam\_kerja    def tampilkan\_info(self):  return f"{super().tampilkan\_info()}, Jam Kerja: {self.jam\_kerja}"  class ManajemenKaryawan:  def \_\_init\_\_(self):  self.daftar\_karyawan = []    def tambah\_karyawan(self, karyawan):  self.daftar\_karyawan.append(karyawan)  def tampilkan\_semua\_karyawan(self):  for karyawan in self.daftar\_karyawan:  print(karyawan.tampilkan\_info())    manajemen = ManajemenKaryawan()  karyawan\_tetap\_1 = KaryawanTetap("Adytta", 5000000, "HR", 1000000)  karyawan\_tetap\_2 = KaryawanTetap("Fauzi", 6000000, "Finance", 1200000)  karyawan\_harian\_1 = KaryawanHarian("Teddy", 3000000, "Marketing", 8)  karyawan\_harian\_2 = KaryawanHarian("Abdul", 3500000, "Sales", 7)  manajemen.tambah\_karyawan(karyawan\_tetap\_1)  manajemen.tambah\_karyawan(karyawan\_tetap\_2)  manajemen.tambah\_karyawan(karyawan\_harian\_1)  manajemen.tambah\_karyawan(karyawan\_harian\_2)  manajemen.tampilkan\_semua\_karyawan() |

#### Source Code Soal 2

|  |
| --- |
| class Pengiriman:  def \_\_init\_\_(self, asal, tujuan):  self.asal = asal  self.tujuan = tujuan  def estimasi\_waktu(self):  return 6  class PengirimanDarat(Pengiriman):  def \_\_init\_\_(self, asal, tujuan, jenis\_kendaraan):  super().\_\_init\_\_(asal, tujuan)  self.jenis\_kendaraan = jenis\_kendaraan  def estimasi\_waktu(self):  if self.jenis\_kendaraan == "mobil":  return 5  elif self.jenis\_kendaraan == "truk":  return 7  else:  return 10  class PengirimanUdara(Pengiriman):  def \_\_init\_\_(self, asal, tujuan, maskapai):  super().\_\_init\_\_(asal, tujuan)  self.maskapai = maskapai  def estimasi\_waktu(self):  if self.maskapai == "Maskapai A":  return 2  elif self.maskapai == "Maskapai B":  return 3  else:  return 4  class PengirimanInternasional(PengirimanDarat, PengirimanUdara):  def \_\_init\_\_(self, asal, tujuan, jenis\_kendaraan, maskapai):  Pengiriman.\_\_init\_\_(self, asal, tujuan)  self.jenis\_kendaraan = jenis\_kendaraan  self.maskapai = maskapai  def estimasi\_waktu(self):  waktu\_darat = PengirimanDarat.estimasi\_waktu(self)  waktu\_udara = PengirimanUdara.estimasi\_waktu(self)  estimasi\_awal = max(waktu\_darat, waktu\_udara)  if self.tujuan not in ["Jakarta", "Bandung", "Surabaya", "Medan"]:  estimasi\_awal += 3  return f"Estimasi waktu pengiriman internasional: {estimasi\_awal} hari"  pengiriman1 = PengirimanInternasional("Jakarta", "Amerika", "mobil", "Maskapai A")  pengiriman2 = PengirimanInternasional("Bandung", "New York", "truk", "Maskapai B")  pengiriman3 = PengirimanInternasional("Surabaya", "London", "motor", "Maskapai C")  print(pengiriman1.estimasi\_waktu())  print(pengiriman2.estimasi\_waktu())  print(pengiriman3.estimasi\_waktu()) |

### Hasil

#### Hasil Soal 1

#### Hasil Soal 2

### Penjelasan

#### 4.4.1 Penjelasan Soal 1

Kode yang saya buat, terdapat beberapa kelas yang digunakan untuk mengelola data karyawan. Kelas Karyawan merupakan kelas dasar dengan atribut nama, gaji, dan departemen, serta metode tampilkan\_info untuk menampilkan informasi karyawan. Kelas KaryawanTetap dan KaryawanHarian adalah turunan dari Karyawan, dengan penambahan atribut spesifik masing-masing, yaitu tunjangan untuk karyawan tetap dan jam\_kerja untuk karyawan harian. Kelas ManajemenKaryawan digunakan untuk mengelola daftar karyawan, dengan metode untuk menambahkan karyawan dan menampilkan seluruh data karyawan.

#### Penjelasan Soal 2

Kode yang saya buat diatas menggunakan metode Kode yang saya buat berisi beberapa kelas yang digunakan untuk menghitung estimasi waktu pengiriman barang berdasarkan moda transportasi dan rute pengirimannya. Kelas Pengiriman adalah kelas dasar yang memiliki atribut asal dan tujuan, serta metode estimasi\_waktu yang mengembalikan nilai default 6 hari. Kelas PengirimanDarat dan PengirimanUdara adalah turunan dari kelas Pengiriman.

## BAB V PENUTUP

### Analisa

Inheritance dan Overriding adalah dua prinsip dasar dalam pemrograman berorientasi objek yang sangat membantu dalam merancang sistem yang efisien dan terstruktur. Inheritance memungkinkan sebuah kelas untuk mewarisi atribut dan metode dari kelas induk, yang mengurangi redundansi dan memfasilitasi penggunaan kembali kode. Hal ini juga memungkinkan pengembangan kode yang lebih modular, di mana fitur-fitur dasar dapat ditangani dalam kelas induk, dan kelas turunan dapat menambahkan atau memodifikasi perilaku sesuai kebutuhan. Di sisi lain, Overriding memberikan fleksibilitas bagi kelas turunan untuk mengganti implementasi metode yang diwarisi dari kelas induk dengan versi yang lebih spesifik dan sesuai konteks.

### Kesimpulan

Inheritance dan Overriding adalah dua konsep inti dalam pemrograman berorientasi objek yang memungkinkan pengembangan kode yang lebih efisien dan terstruktur. Inheritance memungkinkan penggunaan kembali kode dengan mewarisi atribut dan metode dari kelas induk, sementara Overriding memberikan fleksibilitas untuk mengganti implementasi metode di kelas turunan agar lebih spesifik sesuai kebutuhan. Kombinasi keduanya membuat sistem menjadi lebih modular, fleksibel, dan mudah untuk dikembangkan atau disesuaikan, tanpa perlu menulis ulang kode yang sudah ada.

1. Inheritance mengurangi duplikasi kode dengan mewarisi atribut dan metode dari kelas induk, serta memudahkan pemeliharaan kode.
2. Overriding memberi kesempatan pada kelas turunan untuk mengganti implementasi metode dari kelas induk, menyesuaikan perilaku sesuai kebutuhan spesifik.
3. Kombinasi Inheritance dan Overriding menciptakan sistem yang lebih modular, fleksibel, dan mudah dikembangkan tanpa menulis ulang fungsionalitas yang telah ada.

**LAPORAN RESMI**

# MODUL IV Encapsulation + Property

**PEMROGRAMAN BERBASIS OBJEK**

**NAMA : ADYTTA PUTRA TARIGAN**

**N.R.P : 240441100139**

**DOSEN : YUDHA DWI PUTRA NEGARA, S.KOM., M.KOM.**

**ASISTEN : AHMAD RIKHAN ARBA’I**

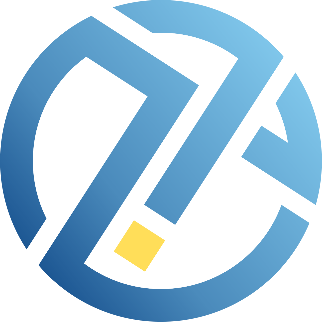
**TGL PRAKTIKUM : 17 MEI 2025**

**Disetujui : MEI 2025**

**Asisten**

**AHMAD RIKHAN ARBA’I**

**23.04.411.00192**



**LABORATORIUM TEKNOLOGI INFORMASI**

**PRODI SISTEM INFORMASI**

**JURUSAN TEKNIK INFORMATIKA**

**FAKULTAS TEKNIK**

**UNIVERSITAS TRUNOJOYO MADURA**

## BAB I PENDAHULUAN

### Latar Belakang

Dalam dunia pemrograman berorientasi objek (OOP), encapsulation adalah konsep yang membatasi akses ke atribut dan metode dalam sebuah objek agar data tetap aman dan hanya dapat diubah sesuai aturan tertentu. Tiga tingkat akses utama—public, protected, dan private—memberikan kontrol terhadap visibilitas dan manipulasi data. Sementara public memberikan akses dari mana saja, protected membatasi akses ke dalam kelas dan turunannya, sedangkan private hanya bisa diakses dari dalam kelas itu sendiri, mencegah perubahan langsung yang tidak diinginkan.

Selain itu, konsep property dengan getter dan setter menjaga kontrol terhadap data, memastikan nilai diambil dan ditetapkan sesuai aturan khusus. Dalam Python, dekorator @property digunakan untuk mendefinisikan getter, sedangkan @property.setter digunakan untuk setter, memberikan cara yang lebih aman dalam mengelola atribut tanpa akses langsung. Dengan kombinasi encapsulation dan property, kode lebih aman, modular, serta mudah dipelihara.

### Tujuan

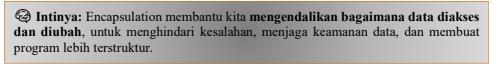
* Mahasiswa mampu memahami konsep *Encapsulation* dalam pemrograman berorientasi objek, serta fungsinya dalam menyembunyikan data untuk menjaga keamanan dan keteraturan program.
* Mahasiswa dapat membedakan jenis akses atribut dan method dalam Python, yaitu *public*, *protected*, dan *private*.
* Mahasiswa mampu menggunakan *property*, serta mengimplementasikan method *getter* dan *setter* untuk mengatur dan mengakses atribut private dengan aman.
* Mahasiswa memahami cara penggunaan @property dan @setter untuk mengontrol logika validasi data pada sebuah *class*.

## BAB II DASAR TEORI

### Pengertian Encapsulation

Apa sih Encapsulation itu??

*Encapsulation* (*enkapsulasi*) adalah salah satu pilar utama dalam *Pemrograman Berorientasi Objek* (*OOP*). Tujuan utama *encapsulation* adalah untuk menyembunyikan data internal objek agar tidak bisa diakses atau dimodifikasi secara langsung dari luar *class*. Alih-alih, data tersebut hanya dapat diakses atau dimodifikasi melalui *method* khusus.



#### Mengapa Encapsulation Penting

1. Keamanan Data

Atribut penting bisa disembunyikan agar tidak sembarangan diubah dari luar.

1. Kontrol Penuh Terhadap Akses

Kita bisa menentukan siapa yang bisa membaca atau mengubah suatu nilai (dengan *getter* dan *setter*).

1. Kemudahan Memelihara Kode

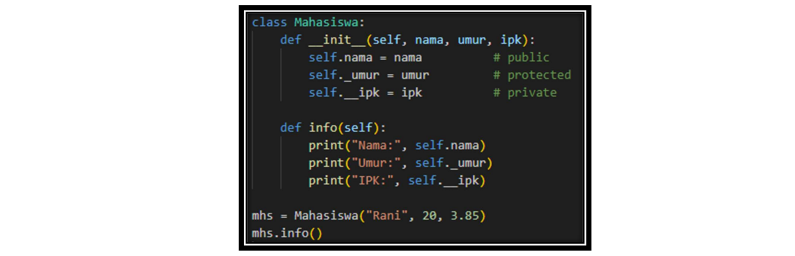
Jika suatu logika validasi harus diubah, cukup di satu tempat (*setter*), tanpa menyentuh seluruh kode.

### Tingkatan Akses di Phyton

Dalam Python, sebenarnya tidak ada keyword akses eksplisit seperti *public, private,* atau *protected* (seperti di Java atau C++). Namun, Python menggunakan konvensi penamaan atribut untuk menunjukkan tingkat aksesibilitas:



### Contoh Kode: Encapsulation Dasar



Penjelasan:

1. self.nama: atribut *public*, bisa diakses bebas.
2. self.\_umur: atribut *protected*, tetap bisa diakses, tapi konvensi menyarankan

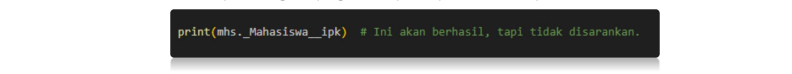
hanya digunakan internal.

1. self.\_\_ipk: *private*, tidak bisa langsung diakses dari luar.

Jika mencoba:



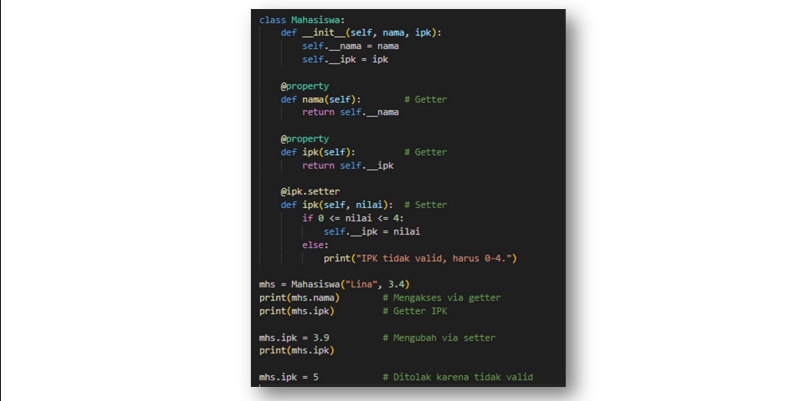
Namun Python tetap menyimpan nilainya, hanya nama atributnya diubah:



### Property: Getter dan Setter

Meneruskan dari sebelumnya, dikarenakan atribut \_\_ipk tidak bisa diakses langsung, bagaimana kita bisa melihat atau mengubahnya? Jawabannya: gunakan *property*, yaitu cara membuat method getter dan setter agar kita bisa mengakses *private attribute* seperti atribut biasa, tetapi tetap melewati logika validasi.

Contoh Penggunaan @property:



Penjelasan:

1. @*property* menjadikan method seperti ipk() bisa dipanggil seperti mhs.ipk.
2. @ipk.setter menetapkan method yang akan dipanggil saat kita mengatur nilai dengan mhs.ipk = nilai.
3. Dengan ini, kita bisa tetap mengontrol perubahan data tanpa melepaskan aksesibilitasnya.

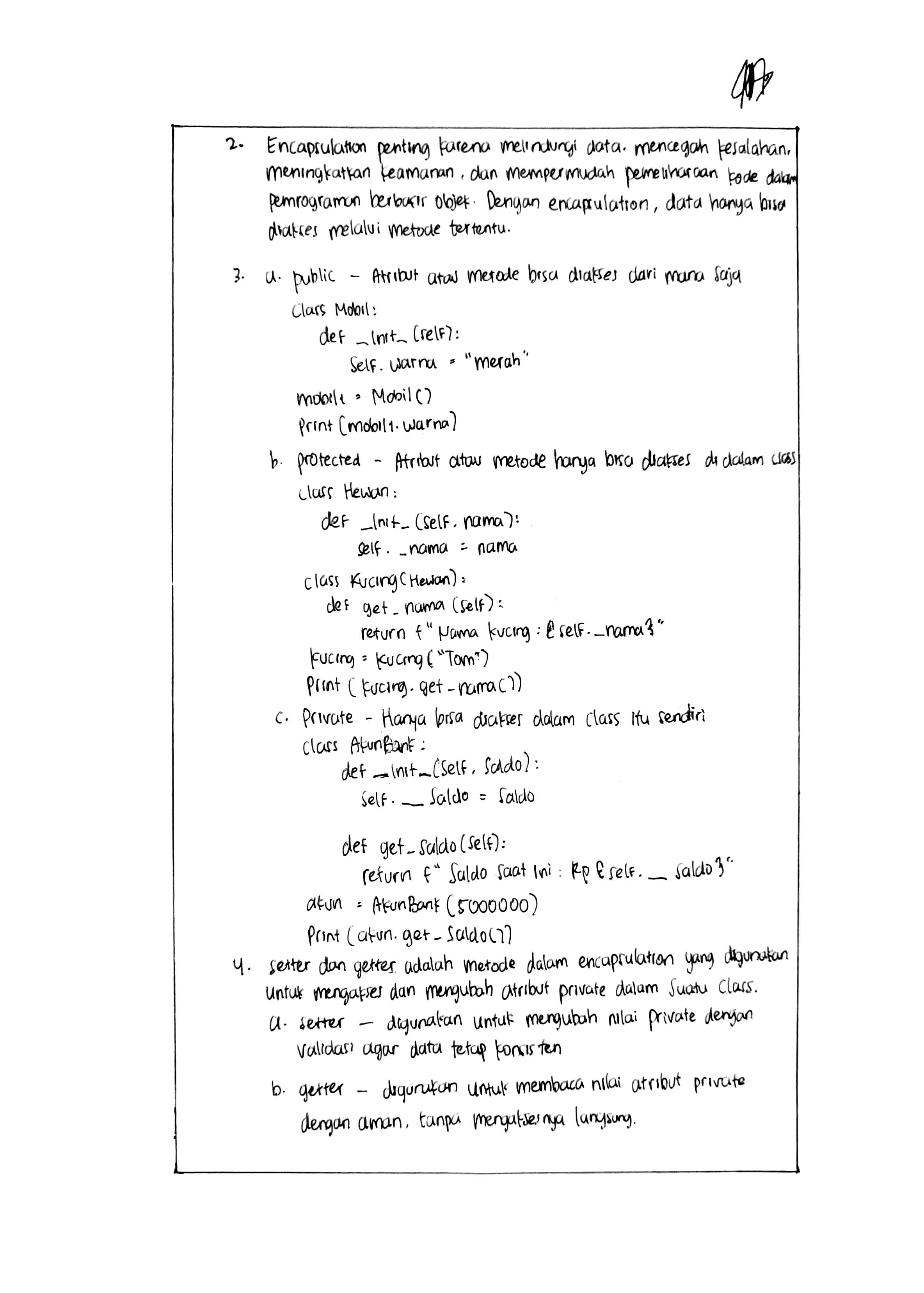
### Studi Kasus Mini – Data Gaji Karyawan

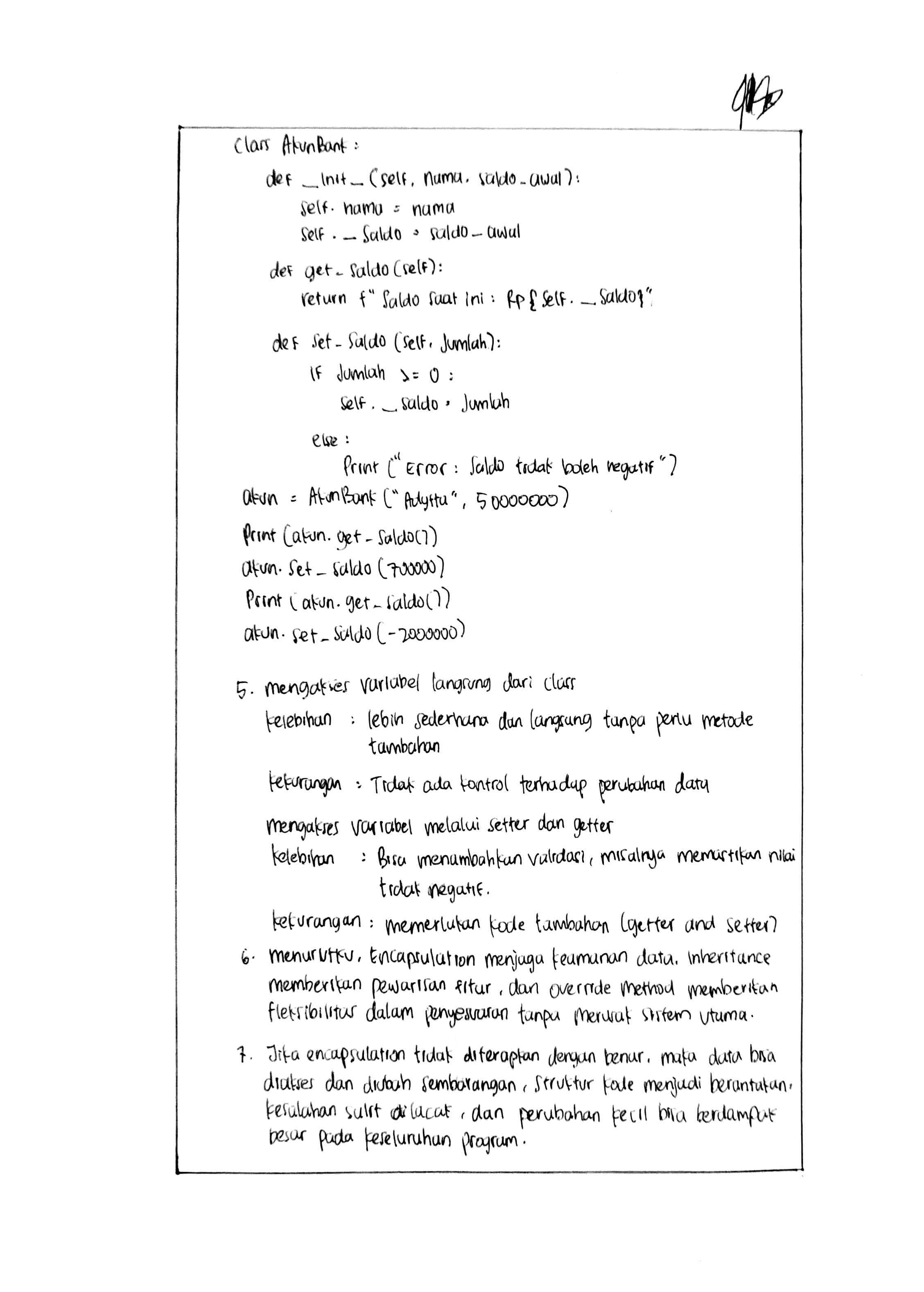


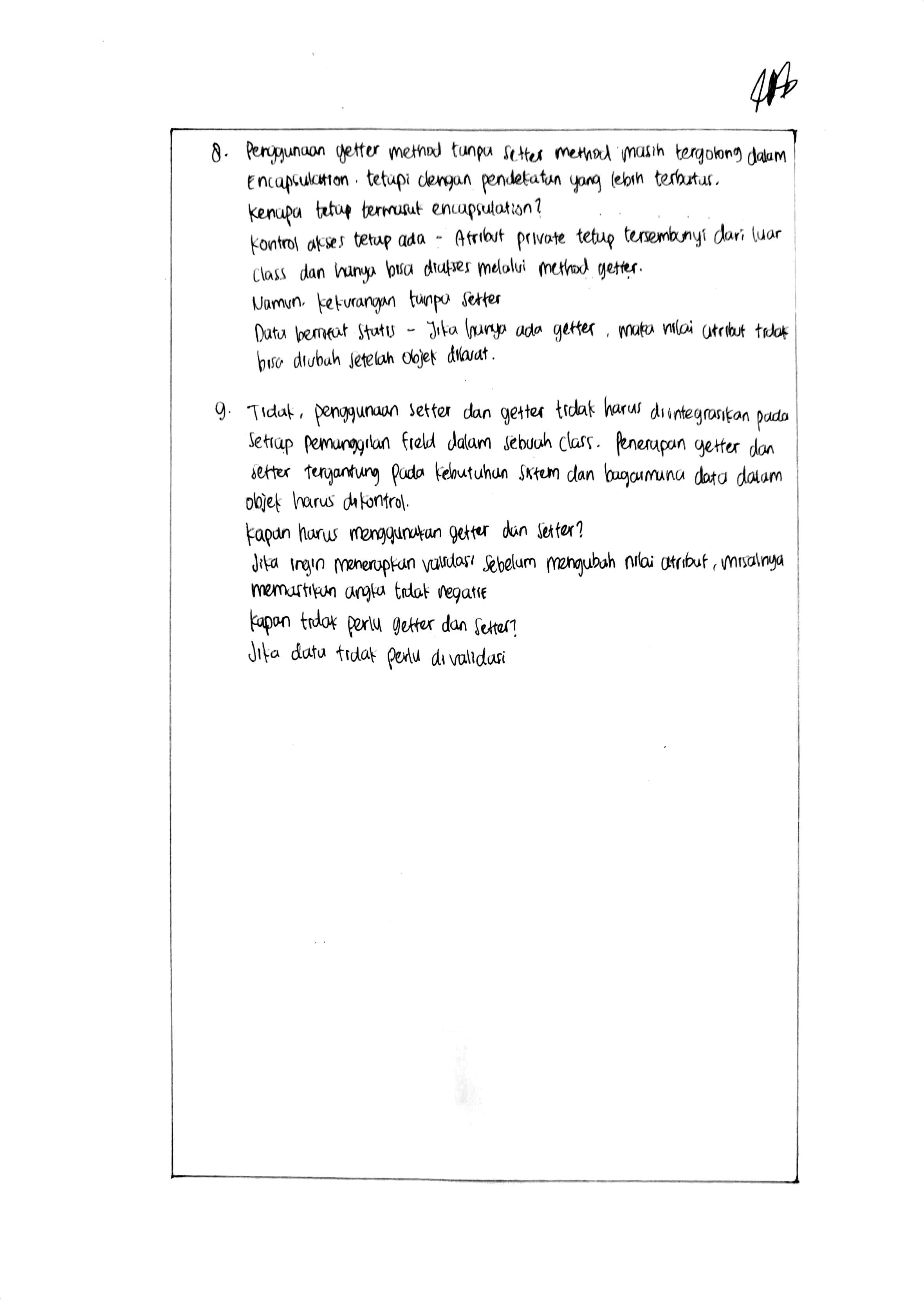
Kesimpulan:

1. *Encapsulation* melindungi data internal agar tidak sembarangan diakses atau dimodifikasi.
2. Python menggunakan penamaan atribut *(\_protected, \_\_private)* untuk menunjukkan tingkat akses.
3. *@property* memungkinkan kita menulis kode yang bersih, tetapi tetap aman, dengan kontrol penuh atas *getter* dan *setter.*

## BAB III TUGAS PENDAHULUAN







## BAB IV IMPLEMENTASI

### Tugas Praktikum

#### Tugas Praktikum No. 1

Sebuah bank punya sistem internal yang nantinya akan menyimpan data rekening setiap nasabah. Setiap nasabah punya no rek, nama pemilik, dan saldo yang nantinya cuma bisa diakses dengan method tertentu.Bank dapat menambahkan rekening, melakukan setran dan penarikan berdasar no rek. Silahkan pakai array untuk nyimpen data rekening. ( Buat class RekeningBank, bank, dan main untuk run file dan menampilkan semua data rekening )

#### Tugas Praktikum No. 2

Perpustakaan ingin mencatat daftar buku yang terdiri dari judul, penulis, dan jumlah halaman. Semua atribut buku harus bersifat private dan hanya bisa diakses dari method tertentu. Silahkan gunakan array untuk menyimpan buku. ( Buat class Buku, Perpustakaan, main untuk run file dan menjalankan beberapa data buku. Penambahan data dan penampilan data buku ditaruh di class Perpustakaan )

#### Tugas Praktikum No. 3

Sebuah Klinik ingin mencatat data pasien dari nama, umur, dan keluhan. Semua data harus bersifat privat dan hanya bisa diakses oleh method tertentu. Klinik dapat menambah pasien dan menampilkan seluruh daftar pasien. ( Buat class Pasien, Klinik, dan main )

### Source Code

#### Source Code Soal 1

|  |
| --- |
| class RekeningBank:  def \_\_init\_\_(self, no\_rekening, nama\_pemilik, saldo):  self.\_\_no\_rekening = no\_rekening  self.\_\_nama\_pemilik = nama\_pemilik  self.\_\_saldo = saldo  @property  def no\_rekening(self):  return self.\_\_no\_rekening  @property  def nama\_pemilik(self):  return self.\_\_nama\_pemilik  @property  def saldo(self):  return self.\_\_saldo  @saldo.setter  def saldo(self, jumlah):  if jumlah >= 0:  self.\_\_saldo = jumlah  else:  print("Saldo tidak boleh negatif!")  def setor\_saldo(self, jumlah):  if jumlah > 0:  self.saldo += jumlah  print(f"Setoran sebesar {jumlah} berhasil.")  else:  print("Jumlah setoran harus lebih dari 0.")  def tarik\_saldo(self, jumlah):  if 0 < jumlah <= self.saldo:  self.saldo -= jumlah  print(f"Penarikan sebesar {jumlah} berhasil.")  else:  print("Saldo tidak mencukupi atau jumlah tidak valid.")  def tampilkan\_info(self):  print(f"No Rekening: {self.no\_rekening}, Nama: {self.nama\_pemilik}, Saldo: {self.saldo}")  class Bank:  def \_\_init\_\_(self):  self.rekening\_list = []  def cari\_rekening(self, no\_rekening):  for rekening in self.rekening\_list:  if rekening.no\_rekening == no\_rekening:  return rekening  return None  def tambah\_rekening(self, rekening):  if self.cari\_rekening(rekening.no\_rekening):  print("Nomor rekening sudah terdaftar.")  else:  self.rekening\_list.append(rekening)  print(f"Rekening atas nama {rekening.nama\_pemilik} berhasil ditambahkan.")  def hapus\_rekening(self, no\_rekening):  rekening = self.cari\_rekening(no\_rekening)  if rekening:  self.rekening\_list.remove(rekening)  print("Rekening berhasil dihapus.")  else:  print("Rekening tidak ditemukan.")  def setor\_saldo(self, no\_rekening, jumlah):  rekening = self.cari\_rekening(no\_rekening)  if rekening:  rekening.setor\_saldo(jumlah)  else:  print("Rekening tidak ditemukan.")  def tarik\_saldo(self, no\_rekening, jumlah):  rekening = self.cari\_rekening(no\_rekening)  if rekening:  rekening.tarik\_saldo(jumlah)  else:  print("Rekening tidak ditemukan.")  def tampilkan\_semua\_rekening(self):  if not self.rekening\_list:  print("Belum ada rekening.")  else:  print("\nData Rekening:")  for rekening in self.rekening\_list:  rekening.tampilkan\_info()  def menu():  bank = Bank()  while True:  print("\n===== MENU BANK =====")  print("1. Tambah Rekening")  print("2. Setor Saldo")  print("3. Tarik Saldo")  print("4. Hapus Rekening")  print("5. Tampilkan Semua Rekening")  print("6. Keluar")  pilihan = input("Pilih menu (1-6): ")  if pilihan == "1":  no\_rek = input("Masukkan No Rekening: ")  nama = input("Masukkan Nama Pemilik: ")  saldo = float(input("Masukkan Saldo Awal: "))  bank.tambah\_rekening(RekeningBank(no\_rek, nama, saldo))  elif pilihan == "2":  no\_rek = input("Masukkan No Rekening: ")  jumlah = float(input("Masukkan Jumlah Setoran: "))  bank.setor\_saldo(no\_rek, jumlah)  elif pilihan == "3":  no\_rek = input("Masukkan No Rekening: ")  jumlah = float(input("Masukkan Jumlah Penarikan: "))  bank.tarik\_saldo(no\_rek, jumlah)  elif pilihan == "4":  no\_rek = input("Masukkan No Rekening yang ingin dihapus: ")  bank.hapus\_rekening(no\_rek)  elif pilihan == "5":  bank.tampilkan\_semua\_rekening()  elif pilihan == "6":  print("Terima kasih telah menggunakan layanan kami.")  break  else:  print("Pilihan tidak valid!")  if \_\_name\_\_ == "\_\_main\_\_":  menu() |

#### Source Code Soal 2

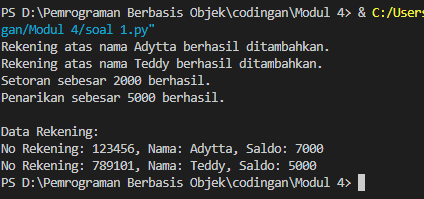
|  |
| --- |
| class Buku:  def \_\_init\_\_(self, judul, penulis, jumlah\_halaman):  self.\_\_judul = judul  self.\_\_penulis = penulis  self.\_\_jumlah\_halaman = jumlah\_halaman  @property  def judul(self):  return self.\_\_judul  @judul.setter  def judul(self, judul\_baru):  self.\_\_judul = judul\_baru  @property  def penulis(self):  return self.\_\_penulis  @penulis.setter  def penulis(self, penulis\_baru):  self.\_\_penulis = penulis\_baru  @property  def jumlah\_halaman(self):  return self.\_\_jumlah\_halaman  @jumlah\_halaman.setter  def jumlah\_halaman(self, halaman\_baru):  self.\_\_jumlah\_halaman = halaman\_baru  def tampilkan\_info(self):  print(f"Judul: {self.judul}, Penulis: {self.penulis}, Halaman: {self.jumlah\_halaman}")  class Perpustakaan:  def \_\_init\_\_(self):  self.daftar\_buku = []  def cari\_buku(self, judul):  for buku in self.daftar\_buku:  if buku.judul.lower() == judul.lower():  return buku  return None  def tambah\_buku(self, buku):  if self.cari\_buku(buku.judul):  print("Buku dengan judul tersebut sudah ada.")  else:  self.daftar\_buku.append(buku)  print(f"Buku '{buku.judul}' berhasil ditambahkan.")  def tampilkan\_semua\_buku(self):  if not self.daftar\_buku:  print("Belum ada buku di perpustakaan.")  else:  print("\nDaftar Buku:")  for buku in self.daftar\_buku:  buku.tampilkan\_info()  def edit\_buku(self, judul):  buku = self.cari\_buku(judul)  if buku:  print("Masukkan data baru (biarkan kosong jika tidak ingin mengubah):")  judul\_baru = input("Judul baru: ").strip()  penulis\_baru = input("Penulis baru: ").strip()  halaman\_baru = input("Jumlah halaman baru: ").strip()  if judul\_baru:  buku.judul = judul\_baru  if penulis\_baru:  buku.penulis = penulis\_baru  if halaman\_baru.isdigit():  buku.jumlah\_halaman = int(halaman\_baru)  elif halaman\_baru:  print("Jumlah halaman harus angka. Data halaman tidak diubah.")  print("Data buku berhasil diperbarui.")  else:  print("Buku tidak ditemukan.")  def pinjam\_buku(self, judul):  buku = self.cari\_buku(judul)  if buku:  self.daftar\_buku.remove(buku)  print(f"Buku '{judul}' berhasil dipinjam.")  else:  print("Buku tidak ditemukan.")  def menu\_perpustakaan():  perpustakaan = Perpustakaan()  while True:  print("\n==== MENU PERPUSTAKAAN ====")  print("1. Tambah Buku")  print("2. Tampilkan Semua Buku")  print("3. Edit Buku")  print("4. Pinjam Buku")  print("5. Keluar")  pilihan = input("Pilih menu (1-5): ")  if pilihan == "1":  judul = input("Masukkan Judul Buku: ").strip()  penulis = input("Masukkan Penulis: ").strip()  halaman = input("Masukkan Jumlah Halaman: ").strip()  if halaman.isdigit():  buku = Buku(judul, penulis, int(halaman))  perpustakaan.tambah\_buku(buku)  else:  print("Jumlah halaman harus berupa angka!")  elif pilihan == "2":  perpustakaan.tampilkan\_semua\_buku()  elif pilihan == "3":  judul = input("Masukkan judul buku yang ingin diedit: ")  perpustakaan.edit\_buku(judul)  elif pilihan == "4":  judul = input("Masukkan judul buku yang ingin dipinjam: ")  perpustakaan.pinjam\_buku(judul)  elif pilihan == "5":  print("Terima kasih sudah menggunakan sistem perpustakaan.")  break  else:  print("Pilihan tidak valid!")  if \_\_name\_\_ == "\_\_main\_\_":  menu\_perpustakaan() |

#### Source Code Soal 3

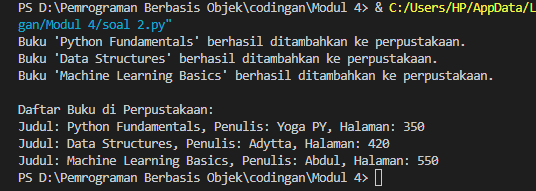
|  |
| --- |
| class Pasien:  def \_\_init\_\_(self, nama, umur, keluhan, ruangan="Belum ditentukan"):  self.\_\_nama = nama  self.\_\_umur = umur  self.\_\_keluhan = keluhan  self.\_\_ruangan = ruangan  @property  def nama(self):  return self.\_\_nama  @nama.setter  def nama(self, nama\_baru):  self.\_\_nama = nama\_baru  @property  def umur(self):  return self.\_\_umur  @umur.setter  def umur(self, umur\_baru):  self.\_\_umur = umur\_baru  @property  def keluhan(self):  return self.\_\_keluhan  @keluhan.setter  def keluhan(self, keluhan\_baru):  self.\_\_keluhan = keluhan\_baru  @property  def ruangan(self):  return self.\_\_ruangan  @ruangan.setter  def ruangan(self, ruangan\_baru):  self.\_\_ruangan = ruangan\_baru  def tampilkan\_info(self):  print(f"Nama: {self.nama}, Umur: {self.umur}, Keluhan: {self.keluhan}, Ruangan: {self.ruangan}")  class Klinik:  def \_\_init\_\_(self):  self.daftar\_pasien = []  def cari\_pasien(self, nama):  for pasien in self.daftar\_pasien:  if pasien.nama.lower() == nama.lower():  return pasien  return None  def tambah\_pasien(self, pasien):  if self.cari\_pasien(pasien.nama):  print("Pasien dengan nama tersebut sudah terdaftar.")  else:  self.daftar\_pasien.append(pasien)  print(f"Pasien '{pasien.nama}' berhasil ditambahkan.")  def tampilkan\_semua\_pasien(self):  if not self.daftar\_pasien:  print("Belum ada pasien terdaftar.")  else:  print("\nDaftar Pasien:")  for pasien in self.daftar\_pasien:  pasien.tampilkan\_info()  def edit\_data\_pasien(self, nama):  pasien = self.cari\_pasien(nama)  if pasien:  print("Biarkan kosong jika tidak ingin mengubah.")  nama\_baru = input("Nama baru: ").strip()  umur\_baru = input("Umur baru: ").strip()  keluhan\_baru = input("Keluhan baru: ").strip()  if nama\_baru:  pasien.nama = nama\_baru  if umur\_baru.isdigit():  pasien.umur = int(umur\_baru)  elif umur\_baru:  print("Umur harus berupa angka. Data umur tidak diubah.")  if keluhan\_baru:  pasien.keluhan = keluhan\_baru  print("Data pasien berhasil diperbarui.")  else:  print("Pasien tidak ditemukan.")  def edit\_ruangan\_pasien(self, nama):  pasien = self.cari\_pasien(nama)  if pasien:  ruangan\_baru = input(f"Masukkan ruangan baru untuk {pasien.nama}: ").strip()  if ruangan\_baru:  pasien.ruangan = ruangan\_baru  print("Ruangan pasien berhasil diperbarui.")  else:  print("Input ruangan kosong. Tidak diubah.")  else:  print("Pasien tidak ditemukan.")  def hapus\_pasien(self, nama):  pasien = self.cari\_pasien(nama)  if pasien:  self.daftar\_pasien.remove(pasien)  print(f"Pasien '{nama}' berhasil dihapus.")  else:  print("Pasien tidak ditemukan.")  # === FITUR BARU: Cari pasien berdasarkan ruangan ===  def cari\_pasien\_berdasarkan\_ruangan(self, ruangan):  ditemukan = False  print(f"\nPasien di ruangan '{ruangan}':")  for pasien in self.daftar\_pasien:  if pasien.ruangan.lower() == ruangan.lower():  pasien.tampilkan\_info()  ditemukan = True  if not ditemukan:  print("Tidak ada pasien di ruangan tersebut.")  def menu\_klinik():  klinik = Klinik()  while True:  print("\n=== MENU KLINIK ===")  print("1. Tambah Pasien")  print("2. Tampilkan Semua Pasien")  print("3. Edit Data Pasien")  print("4. Edit Ruangan Pasien")  print("5. Hapus Pasien")  print("6. Cari Pasien Berdasarkan Ruangan")  print("7. Keluar")  pilihan = input("Pilih menu (1-7): ")  if pilihan == "1":  nama = input("Masukkan nama pasien: ").strip()  umur = input("Masukkan umur pasien: ").strip()  keluhan = input("Masukkan keluhan pasien: ").strip()  ruangan = input("Masukkan ruangan pasien: ").strip()  if umur.isdigit():  pasien = Pasien(nama, int(umur), keluhan, ruangan)  klinik.tambah\_pasien(pasien)  else:  print("Umur harus berupa angka!")  elif pilihan == "2":  klinik.tampilkan\_semua\_pasien()  elif pilihan == "3":  nama = input("Masukkan nama pasien yang ingin diedit: ").strip()  klinik.edit\_data\_pasien(nama)  elif pilihan == "4":  nama = input("Masukkan nama pasien yang ingin diubah ruangannya: ").strip()  klinik.edit\_ruangan\_pasien(nama)  elif pilihan == "5":  nama = input("Masukkan nama pasien yang ingin dihapus: ").strip()  klinik.hapus\_pasien(nama)  elif pilihan == "6":  ruangan = input("Masukkan nama ruangan yang ingin dicari: ").strip()  klinik.cari\_pasien\_berdasarkan\_ruangan(ruangan)  elif pilihan == "7":  print("Terima kasih telah menggunakan sistem klinik.")  break  else:  print("Pilihan tidak valid!")  if \_\_name\_\_ == "\_\_main\_\_":  menu\_klinik() |

### Hasil

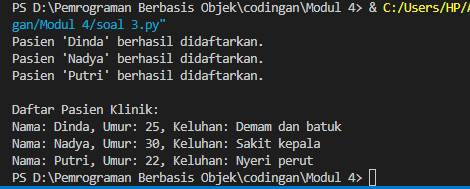
#### Hasil Soal 1



#### Hasil Soal 2



#### Hasil Soal 3



### Penjelasan

#### Penjelasan Soal 1

Kode yang kamu buat terdiri dari dua kelas utama yaitu `RekeningBank` dan `Bank`, serta sebuah fungsi `menu()` untuk menjalankan antarmuka berbasis teks. Kelas `RekeningBank` berfungsi untuk merepresentasikan satu rekening bank dengan atribut privat seperti nomor rekening, nama pemilik, dan saldo. Akses ke atribut-atribut ini dilakukan melalui properti (`@property`) agar tetap terjaga keamanannya. Untuk atribut saldo, digunakan setter yang membatasi agar nilainya tidak bisa diatur ke angka negatif. Selain itu, terdapat metode `setor\_saldo` dan `tarik\_saldo` untuk menangani transaksi, masing-masing disertai validasi agar transaksi sesuai aturan.

Kelas `Bank` bertugas mengelola daftar seluruh rekening yang disimpan dalam atribut `rekening\_list`. Di dalamnya terdapat metode `tambah\_rekening`, `hapus\_rekening`, dan `cari\_rekening` untuk menangani pengelolaan rekening nasabah. Saat nasabah ingin menyetor atau menarik uang, metode `setor\_saldo` dan `tarik\_saldo` digunakan setelah rekening ditemukan terlebih dahulu dengan `cari\_rekening`. Jika nomor rekening tidak ditemukan, sistem akan menampilkan pesan kesalahan.

Fungsi `menu()` menyediakan antarmuka berbasis teks yang interaktif, menampilkan daftar pilihan seperti menambah rekening, menyetor uang, menarik saldo, dan menampilkan seluruh data rekening. Fungsi ini akan terus berjalan hingga pengguna memilih keluar dari program. Setiap input pengguna akan diarahkan ke metode yang sesuai di dalam kelas `Bank`, sehingga seluruh aktivitas transaksi dan pengelolaan rekening terpusat dengan rapi.

#### Penjelasan Soal 2

Kode di atas yang saya buat terdiri dari dua kelas, yaitu Buku dan Perpustakaan. Kelas Buku merepresentasikan sebuah buku dengan atribut privat: judul, penulis, dan jumlah halaman. Getter dengan @property digunakan untuk mengakses data secara aman, dan method tampilkan\_info() digunakan untuk menampilkan detail buku secara rapi.

Sementara itu, kelas Perpustakaan berfungsi untuk menyimpan dan mengelola kumpulan objek buku. Method tambah\_buku() menambahkan buku ke dalam daftar, dan tampilkan\_semua\_buku() akan mencetak semua buku yang ada. Pada bagian if \_\_name\_\_ == "\_\_main\_\_" digunakan untuk mendemonstrasikan penggunaan program dengan menambahkan beberapa buku dan menampilkannya ke konsol. Dengan pendekatan ini, data buku dapat dikelola secara terstruktur dan mudah dikembangkan lebih lanjut, seperti menambahkan fitur pencarian atau penghapusan buku.

#### Penjelasan Soal 3

Kode di atas yang saya buat terdiri dari dua kelas, yaitu Pasien dan Klinik. Kelas Pasien digunakan untuk merepresentasikan data pasien dengan atribut privat: nama, umur, dan keluhan. Akses ke atribut tersebut dilakukan melalui getter @property, dan method tampilkan\_info() berfungsi untuk menampilkan informasi pasien secara lengkap.

Sementara itu, kelas Klinik bertugas mengelola daftar pasien. Method tambah\_pasien() digunakan untuk menambahkan pasien ke dalam daftar, sedangkan tampilkan\_semua\_pasien() akan mencetak seluruh data pasien yang telah terdaftar. Pada bagian akhir, program mendemonstrasikan cara kerja kelas dengan membuat tiga objek pasien dan menampilkannya di konsol. Dengan pendekatan ini, data pasien bisa dicatat, ditampilkan, dan dikelola secara terorganisir sesuai kebutuhan klinik.

## BAB V PENUTUP

### Analisa

Analisis dalam konteks ini mengacu pada pemahaman bagaimana encapsulation dan property mempengaruhi struktur dan keamanan data dalam pemrograman berorientasi objek. Dengan membatasi akses menggunakan *public, protected*, dan *private*, serta menerapkan *getter* dan *setter*, pengembang dapat memastikan bahwa data dalam objek hanya diubah melalui mekanisme yang telah ditentukan. Hal ini mengurangi risiko kesalahan akibat manipulasi langsung dan meningkatkan keteraturan kode. Analisis ini juga menunjukkan bahwa prinsip-prinsip tersebut berperan penting dalam membangun sistem yang modular, aman, dan mudah dipelihara, sehingga mempermudah pengembangan perangkat lunak dalam jangka panjang.

### Kesimpulan

*Encapsulation* dan *property* berperan penting dalam menjaga keamanan dan keteraturan data dalam pemrograman berorientasi objek. Dengan membatasi akses atribut serta menerapkan *getter* dan *setter*, pengembang dapat memastikan kontrol yang lebih baik terhadap data, mencegah manipulasi yang tidak diinginkan, dan meningkatkan modularitas sistem. Penerapan konsep ini membantu membangun kode yang lebih rapi, aman, serta mudah dipelihara, sehingga mendukung pengembangan perangkat lunak yang lebih efisien dan berkelanjutan

* 1. Encapsulation membatasi akses ke atribut dan metode (*public*, *protected*, *private*) untuk menjaga keamanan dan keteraturan data dalam suatu objek.
  2. *Property* dengan *getter* dan *setter* memastikan bahwa data hanya dapat diubah melalui mekanisme yang terkontrol, mencegah manipulasi langsung.
  3. Kombinasi kedua konsep ini membantu menciptakan sistem yang lebih aman, modular, dan mudah dipelihara dalam pemrograman berorientasi objek.

**LAPORAN RESMI**

# MODUL V ABSTRACT CLASS & ABSTRACT METHOD

**PEMROGRAMAN BERBASIS OBJEK**

**NAMA : ADYTTA PUTRA TARIGAN**

**N.R.P : 240441100139**

**DOSEN : YUDHA DWI PUTRA NEGARA, S.KOM., M.KOM.**

**ASISTEN : AHMAD RIKHAN ARBA’I**

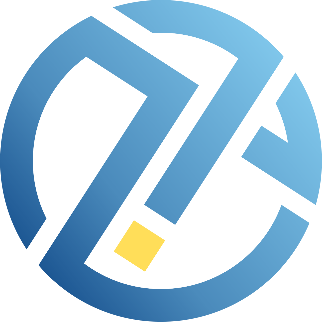
**TGL PRAKTIKUM : 17 MEI 2025**

**Disetujui : 28 MEI 2025**

**Asisten**

**AHMAD RIKHAN ARBA’I**

**23.04.411.00192**



**LABORATORIUM TEKNOLOGI INFORMASI**

**PRODI SISTEM INFORMASI**

**JURUSAN TEKNIK INFORMATIKA**

**FAKULTAS TEKNIK**

**UNIVERSITAS TRUNOJOYO MADURA**

## BAB I PENDAHULUAN

### Latar Belakang

Dalam pemrograman berorientasi objek, konsep abstrak *class* dan abstrak muncul sebagai solusi untuk menciptakan kerangka dasar dari sebuah sistem yang kompleks. Abstrak *class* merupakan *class* yang tidak bisa diinstansiasi secara langsung, namun dirancang untuk menjadi fondasi bagi *class*-*class* turunannya. Dengan menyediakan struktur umum—seperti atribut dan yang dapat diwariskan—abstrak *class* membantu pengembang menyusun *blueprint* yang konsisten dalam sebuah hierarki objek. Konsep ini sangat berguna dalam sistem besar yang memiliki banyak entitas serupa namun memiliki perilaku yang sedikit berbeda.

Sementara itu, abstrak adalah yang dideklarasikan di dalam abstrak *class* namun tidak memiliki implementasi langsung. Tujuannya adalah untuk memaksa *class* turunan memberikan implementasi spesifik terhadap tersebut. Dengan adanya abstrak , *programmer* dapat menjamin bahwa semua *subclass* akan memiliki fungsi-fungsi penting tertentu yang telah didefinisikan secara abstrak di level atas. Hal ini memberikan kontrol dan konsistensi dalam desain sistem, sekaligus fleksibilitas dalam implementasi. Oleh karena itu, penggunaan abstrak dan abstrak *class* menjadi elemen penting dalam membangun sistem perangkat lunak yang modular, skalabel, dan mudah dikembangkan ke depannya.

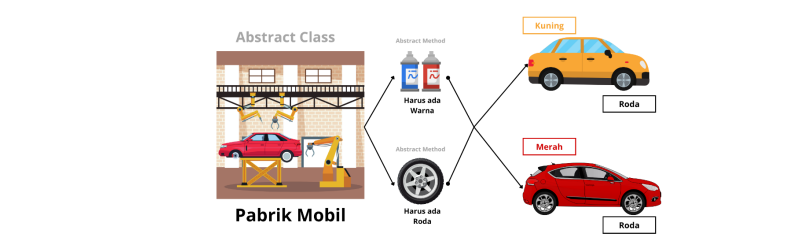
### Tujuan

* Mahasiswa mampu memahami konsep *abstract* *class* dan *abstract* dalam OOP.
* Mahasiswa dapat mengimplementasikan *abstract* *class* menggunakan modul abc di Python.
* Mahasiswa dapat menerapkan konsep tersebut dalam studi kasus sederhana.

## BAB II DASAR TEORI

### *ABSTRACT* *CLASS*

Abstract Class, merupakan suatu Class yang dideklarasikan sebagai abstract, dan merupakan kelas yang berisi satu atau lebih perilaku yang diabstraksi. Yang berarti bahwa objek atau kelas tersebut dapat diringkas menjadi karakteristik yang relevan. Cara kerja abstract class itu ketika di-subclass-kan, subclass tersebut biasanya menyediakan implementasi untuk semua metode abstract di parent class. Namun, jika tidak, maka subclass-nya juga harus dideklarasikan abstract..



Analogi sederhananya seperti, semua kendaraan harus bisa "bergerak", tapi cara bergeraknya bisa berbeda-beda tergantung jenisnya. Blueprint ini tidak bisa digunakan langsung (tidak bisa dibuat kendaraan dari blueprint saja), tapi digunakan sebagai dasar untuk membuat mobil, motor, atau sepeda yang masing-masing memiliki cara bergerak tersendiri.

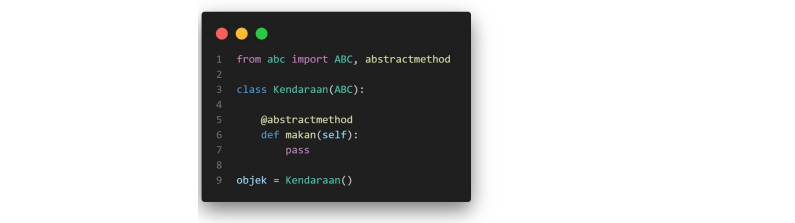
#### Ciri – Ciri dan Tujuan *Abstract* *Class* pada Python

1. *Abstract* *Class* di Python mewarisi kelas ABC dari modul abc



1. Tidak dapat membuat objek dari *abstract* *class*. Jika dicoba, akan menghasilkan *error*.

Code:



Output:



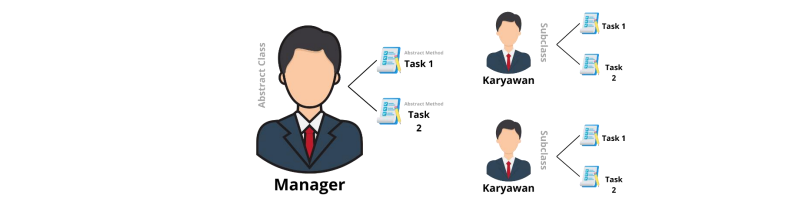
1. Digunakan untuk mendefinisikan struktur dasar yang akan diturunkan ke *subclass*.
2. *Abstract* *class* bisa berisi kombinasi antara biasa (dengan isi) dan *abstract* (tanpa isi).



1. Memastikan semua turunan memiliki struktur dan yang sama meskipun implementasinya berbeda.

### *Abstract*

*Abstract*  adalah (fungsi dalam *class*) yang hanya dideklarasikan tetapi tidak memiliki isi (implementasi) di dalam *class* induknya. ini dibuat menggunakan *decorator* @*abstract*, dan hanya bisa digunakan dalam *abstract class*. Tujuannya adalah untuk memaksa setiap *subclass* agar mengisi atau mengimplementasikan sendiri isi dari tersebut. Dengan begitu, *abstract*  membantu menjaga agar semua *subclass* memiliki struktur yang sama, meskipun perilakunya bisa berbeda-beda sesuai kebutuhan.



Analoginya seperti kontrak kerja: Misalnya seorang manajer menyusun daftar tugas yang harus dikerjakan oleh semua karyawan baru, tapi tidak memberitahu detail cara mengerjakannya—itu terserah karyawan. Yang penting, setiap karyawan harus menyelesaikan tugas yang sama, meskipun dengan cara yang berbeda. Dalam OOP, manajer itu adalah abstract class, daftar tugas adalah abstract , dan karyawan adalah subclass yang mengisi detail cara menyelesaikan tugas tersebut.

#### Ciri – Ciri *Abstract* pada Python

1. Didekorasi dengan @*abstract*

*Abstract* ditandai dengan *decorator @abstract* yang berasal dari modul abc. Ini menandakan bahwa ini belum memiliki implementasi.



1. Tidak memiliki implementasi (hanya deklarasi)

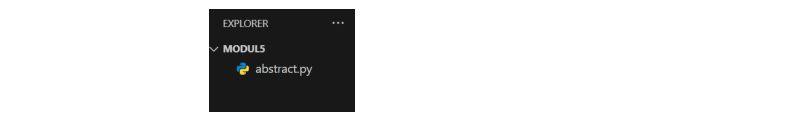
*Abstract* hanya dideklarasikan dengan nama dan parameter, tapi tidak ada logika atau isi di dalamnya. Biasanya menggunakan pass sebagai *placeholder*.



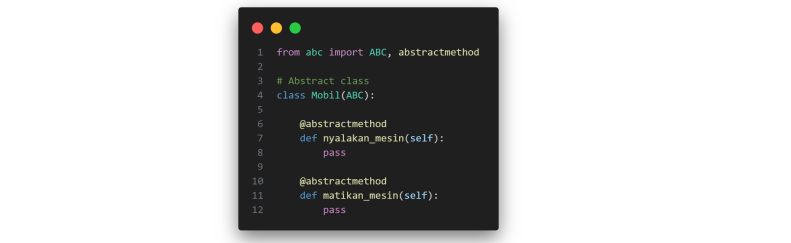
1. Wajib diimplementasikan oleh *subclass*
2. Tidak dapat dipanggil langsung dari *abstract* *class*

### Implementasi *Abstract* *Class* dan *Abstract* pada Program Python

1. Buat folder baru untuk menyimpan project python dan buat file didalamnya.



1. Selanjutnya buat *Abstract* *Class* dengan nama Mobil.



1. Kemudian buat *abstract* pada *class* mobil tadi, dengan nama nyalakan\_mesin() dan matikan\_mesin(). Pastikan pada *abstract* diberi decorator @*abstract*.



1. Kemudian buat *subclass*nya, sebagai contoh buat *subclass* dengan nama MobilSport().



1. Setelah berhasil membuat *class* MobilSport() yang merupakan *subclass* dari *class* Mobil tadi. Tambahkan untuk mengoverride pada *abstract* *class*. (pastikan semua *abstract* sudah diterapkan pada *subclass*nya).



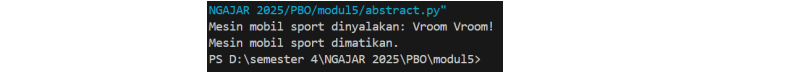
1. Tambahkan aksi dalam nyalakan\_mesin() dan juga matikan\_mesin().



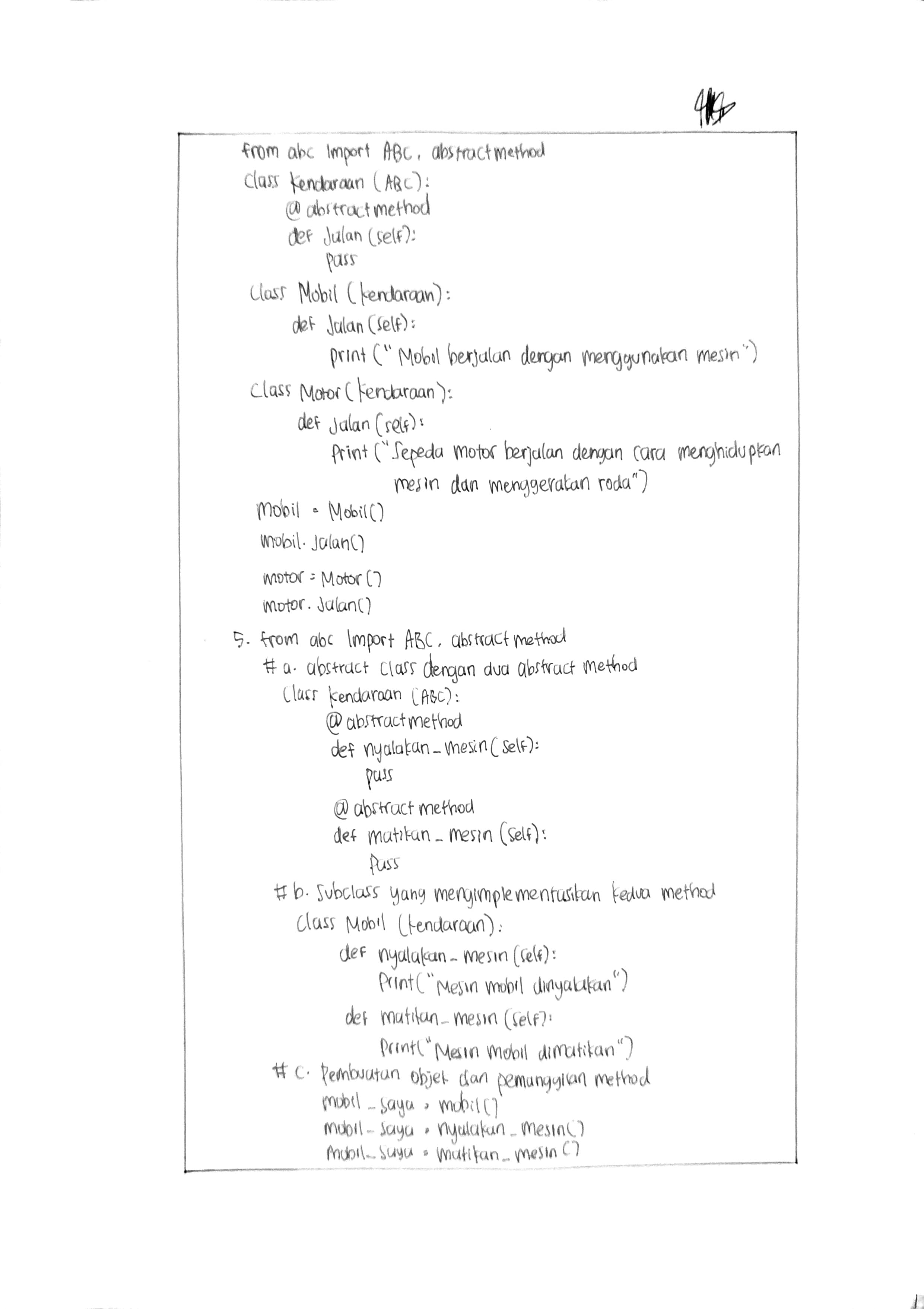
1. Langkah terakhir, buat objek dari MobilSport() yang merupkan *subclass* dari *abstract* *class*.

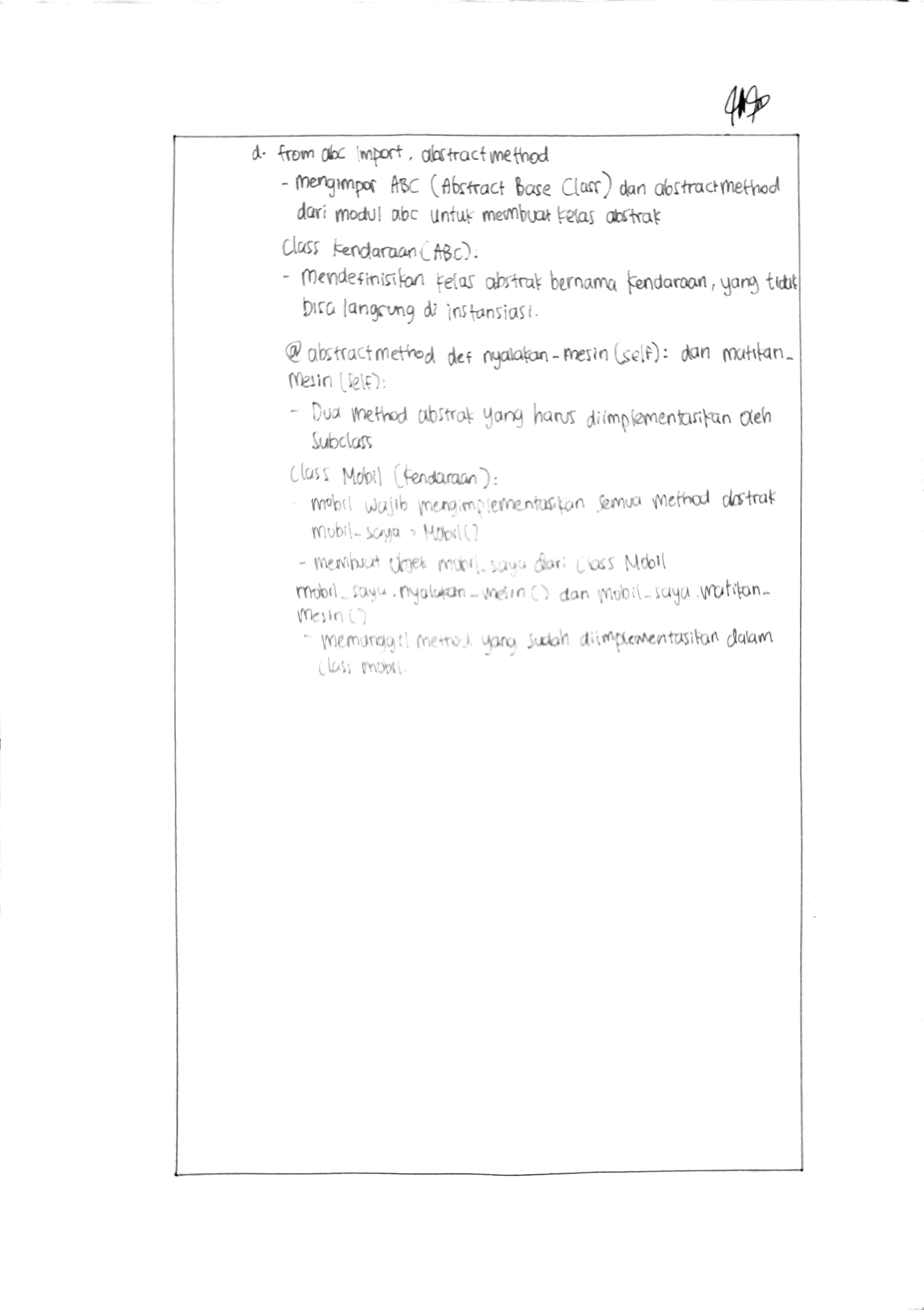


1. Nanti outputnya akan:



## BAB III TUGAS PENDAHULUAN





## BAB IV IMPLEMENTASI

### Tugas Praktikum

#### Tugas Praktikum No. 1

Buatlah sebuah program Python dengan ketentuan sebagai berikut:

1. Buat *abstract* *class* bernama Manusia yang memiliki tiga *abstract* :
   1. berbicara()
   2. bekerja()
   3. makan()
2. Buat empat *subclass* dari Manusia, yaitu: Joko, Beni, Fani, dan Jani. Masing-masing *subclass* mengimplementasikan semua abstrak tersebut dengan pesan berbeda sesuai karakter mereka.
3. Buat objek dari masing-masing *subclass* dan panggil setiap -nya satu per satu (tanpa menggunakan list atau perulangan).

#### Tugas Praktikum No. 2

* + 1. Buatlah sebuah *abstract* *class* bernama PerangkatElektronik dengan:

1. Tiga *abstract* :

* nyalakan() – untuk menyalakan perangkat.
* matikan() – untuk mematikan perangkat.
* gunakan(jam: int) – menerima parameter durasi pemakaian dalam jam.
  + 1. Tambahkan properti:

1. energi\_tersisa bertipe integer.
2. Set nilai awalnya di constructor sebagai 100 (maksimal 100%).
   * 1. Buat dua *subclass* dari PerangkatElektronik:
3. Laptop

* Mengurangi energi\_tersisa sebesar 10 jam setiap kali gunakan() dipanggil.
* Jika energi\_tersisa kurang dari 0, set ke 0 dan cetak pesan bahwa baterai habis.

1. Kulkas

* Mengurangi energi\_tersisa sebesar 5 jam.
* Jika energi turun di bawah 20, cetak peringatan “Energi rendah, kulkas butuh daya tambahan!”
  + 1. Tambahkan status() yang dapat digunakan untuk menampilkan:

1. Tipe perangkat
2. Energi tersisa
   * 1. Buat objek dari masing-masing *subclass*, panggil semua metodenya dengan nilai jam berbeda (tanpa list atau perulangan).

### Source Code

#### Source Code Soal 1

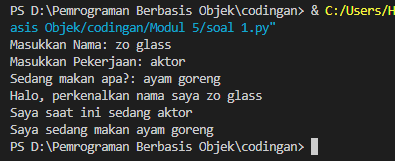
|  |
| --- |
| from abc import ABC, abstractmethod  class Manusia(ABC):  def \_\_init\_\_(self, nama):  self.nama = nama  @abstractmethod  def berbicara(self):  pass  @abstractmethod  def bekerja(self):  pass  @abstractmethod  def makan(self):  pass  class Orang(Manusia):  def \_\_init\_\_(self, nama, pekerjaan, makanan):  super().\_\_init\_\_(nama)  self.pekerjaan = pekerjaan  self.makanan = makanan  def berbicara(self):  print(f"Halo, perkenalkan nama saya {self.nama}")  def bekerja(self):  print(f"Saya saat ini sedang {self.pekerjaan}")  def makan(self):  print(f"Saya sedang makan {self.makanan}")  def main():  nama = input("Masukkan Nama: ")  pekerjaan = input("Masukkan Pekerjaan: ")  makanan = input("Sedang makan apa?: ")  manusia = Orang(nama, pekerjaan, makanan)  manusia.berbicara()  manusia.bekerja()  manusia.makan()  if \_\_name\_\_ == "\_\_main\_\_":  main() |

#### Source Code Soal 2

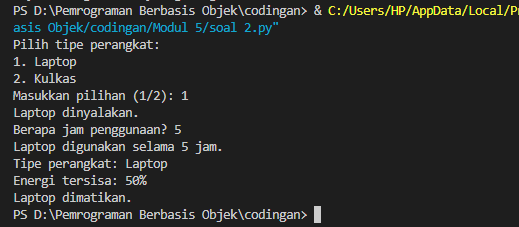
|  |
| --- |
| from abc import ABC, abstractmethod  class PerangkatElektronik(ABC):  def \_\_init\_\_(self, tipe: str): # <-- perbaikan: \_\_init\_\_ bukan \_init\_  self.tipe = tipe  self.energi\_tersisa = 100  @abstractmethod  def nyalakan(self):  pass  @abstractmethod  def matikan(self):  pass  @abstractmethod  def gunakan(self, jam: int):  pass  def status(self):  print(f"Tipe perangkat: {self.tipe}")  print(f"Energi tersisa: {self.energi\_tersisa}%")  class Perangkat(PerangkatElektronik):  def \_\_init\_\_(self, tipe: str):  super().\_\_init\_\_(tipe)  def nyalakan(self):  print(f"{self.tipe} dinyalakan.")  def matikan(self):  print(f"{self.tipe} dimatikan.")  def gunakan(self, jam: int):  if self.tipe == "Laptop":  pemakaian = 10 jam  self.energi\_tersisa -= pemakaian  if self.energi\_tersisa <= 0:  self.energi\_tersisa = 0  print("Baterai habis. Laptop mati.")  else:  print(f"Laptop digunakan selama {jam} jam.")  elif self.tipe == "Kulkas":  pemakaian = 5 jam  self.energi\_tersisa -= pemakaian  if self.energi\_tersisa <= 0:  self.energi\_tersisa = 0  print("Energi habis. Kulkas mati.")  elif self.energi\_tersisa < 20:  print("Energi rendah, kulkas butuh daya tambahan!")  print(f"Kulkas beroperasi selama {jam} jam.")  else:  print("Tipe perangkat tidak dikenali.")  def main():  print("Pilih tipe perangkat:")  print("1. Laptop")  print("2. Kulkas")  pilihan = input("Masukkan pilihan (1/2): ")  if pilihan == "1":  tipe = "Laptop"  elif pilihan == "2":  tipe = "Kulkas"  else:  print("Pilihan tidak valid. Harus 1 atau 2.")  return  perangkat = Perangkat(tipe)  perangkat.nyalakan()  jam\_str = input("Berapa jam penggunaan? ")  if not jam\_str.isdigit():  print("Jam harus berupa angka.")  return  jam = int(jam\_str)  perangkat.gunakan(jam)  perangkat.status()  perangkat.matikan()  if \_\_name\_\_ == "\_\_main\_\_":  main() |

### Hasil

#### Hasil Soal 1



#### Hasil Soal 2



### Penjelasan

#### Penjelasan Soal 1

Codingan yang saya buat di atas menggunakan bahasa Python. Kelas abstrak `Manusia` didefinisikan sebagai *blueprint* dengan tiga metode abstrak, yaitu `berbicara`, `bekerja`, dan `makan`, yang harus diimplementasikan oleh kelas turunannya. Kelas `Orang` merupakan turunan dari `Manusia` yang mengimplementasikan ketiga metode tersebut sesuai dengan atribut `nama`, `pekerjaan`, dan `makanan` yang diberikan saat objek dibuat. Fungsi `main()` berfungsi untuk menerima input dari pengguna dan mencetak hasil implementasi metode-metode yang telah diisi melalui objek `Orang`. Program ini menunjukkan bagaimana pewarisan dan abstraksi dapat digunakan untuk membuat struktur kode yang terorganisir dan fleksibel.

.

#### Penjelasan Soal 2

Codingan yang saya buat di atas merupakan Python untuk mensimulasikan perilaku perangkat elektronik. Kelas abstrak PerangkatElektronik berfungsi sebagai kerangka dasar dengan atribut tipe dan energi\_tersisa, serta tiga metode abstrak: nyalakan, matikan, dan gunakan, yang wajib diimplementasikan oleh kelas turunannya. Kelas Perangkat mengimplementasikan ketiga metode tersebut dengan perilaku berbeda tergantung tipe perangkat, yaitu "Laptop" atau "Kulkas", yang mempengaruhi konsumsi energi per jam. Pada fungsi main(), pengguna diminta memilih jenis perangkat dan durasi penggunaan dalam jam, lalu program menampilkan aktivitas perangkat tersebut seperti dinyalakan, digunakan, status energi, dan dimatikan. Kode ini menunjukkan bagaimana pewarisan dan polymorphism dapat digunakan untuk menangani berbagai jenis perangkat dengan logika penggunaan yang berbeda, sekaligus memberikan interaksi sederhana dengan pengguna melalui *input* dan *output* yang intuitif. Selain itu, kondisi seperti energi habis atau rendah juga ditangani dengan baik untuk menambah *realisme* dalam simulasi.

## BAB V PENUTUP

### Analisa

*Abstract class* berperan sebagai kerangka dasar yang tidak dapat diinstansiasi langsung, melainkan diturunkan oleh *subclass*. Dengan adanya *abstract*  di dalamnya, *abstract class* memastikan bahwa setiap *subclass* memiliki struktur perilaku yang sama meskipun implementasinya berbeda.

Konsep ini sangat penting dalam *object-oriented programming* karena membantu menjaga konsistensi dan memperkuat prinsip pewarisan. Dengan menggunakan *abstract class* dan *abstract* , pengembang dapat membuat sistem yang lebih terstruktur, fleksibel, dan mudah dikembangkan.

### Kesimpulan

Dalam pengembangan perangkat lunak berbasis object-oriented programming, penggunaan abstract class dan abstract menjadi sangat penting untuk menciptakan arsitektur kode yang rapi, konsisten, dan mudah dikembangkan. Abstract class berfungsi sebagai cetak biru yang mendefinisikan struktur dasar tanpa memberikan implementasi penuh, sementara abstract menjadi kontrak yang mewajibkan setiap subclass untuk mengisi sendiri perilaku yang diharapkan. Dengan penerapan konsep ini, pengembang dapat membangun sistem yang fleksibel, menjaga standar struktur antar kelas, serta mempermudah proses pemeliharaan dan pengembangan di masa depan.

1. *Abstract class* tidak dapat diinstansiasi langsung — hanya bisa diwariskan oleh *subclass*.
2. *Abstract*  wajib diimplementasikan oleh *subclass* — karena tidak memiliki isi di kelas induk.
3. Menjaga konsistensi dan struktur dalam pewarisan kode — sehingga semua turunan mengikuti pola yang sama.

**BIOGRAFI PENYUSUN**



**FOT** **O FORMAL**

**Nama** : Adytta Putra Tarigan

**TTL**  : Jakarta, 4 November 2005

**Alamat** : Jl. Rawa Binong Lubang Buaya Cipayung Jakarta Timur

**Jurusan**  : Sistem Informasi

**Universitas** : Universitas Trunojoyo Madura

**Hobi**  : Nongkrong sesame eksmud

**Cita-cita**  : Mau jadi eksmud

**HP**  : 081384103822

**e-Mail**  : [tariganadytta27@gmail.com](mailto:tariganadytta27@gmail.com)

**RIWAYAT PENDIDIKAN**

1. SDN LUBANG BUAYA 03 (2012-2018)
2. SMPN 272 JAKARTA (2018-2021)
3. SMAS PGRI 24 JAKARTA (2021-2024)
4. UNIVERSITAS TRUNOJOYO MADURA (2024-Sekarang)