

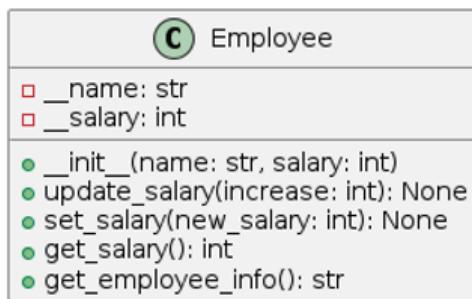


JOBSHEET 03:

KONSTRUKTOR, DESTRUKTOR, DAN ENKAPSULASI

Praktikum 02: Enkapsulasi pada kelas Employee

Berikut ini adalah contoh lain tentang enkapsulasi dalam Python menggunakan kelas Employee yang menyembunyikan data karyawan seperti nama dan gaji, dari akses langsung di luar kelas. Pada contoh ini, kita menyediakan method khusus (getter dan setter) untuk mengakses dan memodifikasi data privat. UML kelas diagram untuk Employee Class.



- **Atribut Privat (-):**
 - `__name: str` → Menyimpan nama karyawan.
 - `__salary: int` → Menyimpan gaji karyawan.
- **Metode Publik (+):**
 - `__init__(name, salary)` → Konstruktor untuk inisialisasi nama dan gaji.
 - `update_salary(increase)` → Menambahkan gaji jika nilai kenaikan valid.
 - `set_salary(new_salary)` → Mengatur ulang gaji dengan validasi.
 - `get_salary()` → Mengembalikan nilai gaji saat ini.
 - `get_employee_info()` → Menampilkan info lengkap tentang karyawan.

Dosen:

Ir. Prayitno, S.ST., M.T., Ph.D.

Nama Mahasiswa : _____
NIM Mahasiswa : _____



Tahun Akademik 2025

A. Tujuan Instruksional Khusus

Setelah menyelesaikan praktikum ini, mahasiswa diharapkan mampu:

1. Menjelaskan dan menggunakan konstruktor (`__init__`) dan destruktor (`__del__`)
2. Memahami peran `self` dalam method sebuah kelas
3. Menerapkan konsep enkapsulasi (encapsulation) dalam Python
4. Menggunakan getter, setter, dan dekorator `@property` untuk mengelola akses atribut

B. Alat dan Bahan

Perangkat Lunak:

- Sistem operasi Windows/Linux/macOS (sesuai ketersediaan).
- Python 3.x (versi terbaru yang stabil).
- IDE atau Code Editor (misalnya, PyCharm, VS Code, atau IDLE).

Perangkat Keras:

- Komputer/Laptop dengan spesifikasi memadai untuk menjalankan Python.

Bahan Pendukung:

- Modul/slide perkuliahan yang menjelaskan dasar pemrograman Python.
- Koneksi internet (opsional, untuk referensi tambahan).

C. Dasar Teori

Pemrograman Berorientasi Objek (Object-Oriented Programming/OOP) merupakan salah satu paradigma pemrograman yang berfokus pada penciptaan objek-objek yang merepresentasikan entitas dunia nyata. Setiap objek merupakan instansi dari sebuah kelas, yang menggabungkan data (disebut atribut) dan perilaku (disebut metode) ke dalam satu unit terstruktur. OOP menawarkan prinsip-prinsip penting seperti enkapsulasi, pewarisan, dan polimorfisme, yang memungkinkan pengembangan perangkat lunak menjadi lebih modular, terorganisir, serta mudah dalam proses pemeliharaan dan pengembangan ulang (Abdul Rauf et al., 2020).

Dalam bahasa pemrograman Python, kelas digunakan sebagai cetakan untuk menciptakan objek. Konstruktor dalam Python direpresentasikan dengan metode khusus `__init__()`, yang dipanggil secara otomatis saat objek dibuat. Konstruktor berfungsi untuk menginisialisasi nilai awal atribut pada objek. Sebaliknya, destruktor (`__del__()`) digunakan untuk membersihkan sumber daya ketika objek tidak lagi digunakan. Walaupun Python memiliki sistem manajemen memori otomatis melalui garbage collector, destruktor tetap bermanfaat terutama untuk menutup koneksi atau file yang dibuka sebelumnya (Ananda & Lestari, 2019).

Kata kunci `self` pada setiap metode dalam kelas Python digunakan untuk merepresentasikan objek yang sedang diproses. `self` memungkinkan setiap objek memiliki atribut dan perilaku yang unik. Tanpa `self`, atribut tidak akan bisa dibedakan antara satu objek dengan objek lainnya. Oleh karena itu, penggunaan `self` merupakan ciri khas yang membedakan metode instance dengan metode statis atau kelas dalam Python (Widodo, 2021).

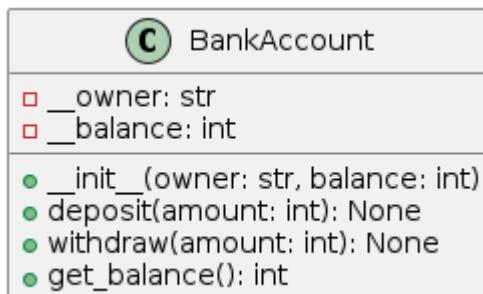
Enkapsulasi, sebagai salah satu pilar OOP, mengacu pada praktik menyembunyikan atribut dari akses langsung oleh pengguna luar. Dalam Python, ini dilakukan dengan menjadikan atribut bersifat privat menggunakan dua garis bawah di depan nama atribut, seperti `_nama`. Untuk mengakses dan memodifikasi atribut privat, digunakan metode **getter** dan **setter**. Dengan dekorator `@property`, Python memungkinkan penulisan getter dan setter yang lebih ringkas dan pythonik. Dekorator ini mengizinkan akses atribut privat seolah-olah mereka adalah atribut publik, tetapi tetap menjaga logika validasi atau kontrol internal. Hal ini memberikan fleksibilitas sekaligus melindungi integritas data pada objek (Suryani et al., 2022).

Implementasi konsep-konsep tersebut sangat berguna dalam pengembangan sistem informasi, aplikasi desktop, maupun layanan berbasis cloud karena memberikan pondasi yang kuat untuk pengorganisasian kode. Pemahaman terhadap prinsip OOP, khususnya dalam pengelolaan objek melalui konstruktor, destruktur, dan pengaturan atribut dengan @property, penting bagi mahasiswa untuk mampu membangun sistem perangkat lunak yang efektif, efisien, dan berkelanjutan.

D. Langkah Praktikum

Praktikum 01: Enkapsulasi pada kelas Bank Account

Berikut adalah contoh sederhana dalam Python yang mendemonstrasikan proses enkapsulasi (encapsulation). Enkapsulasi adalah konsep yang bertujuan untuk “menyembunyikan” atau melindungi data (atribut) di dalam suatu objek agar tidak diakses secara langsung di luar kelas, serta menyediakan metode (fungsi) khusus untuk memanipulasi data tersebut. UML kelas diagram kelas BankAccount.



- **Atribut Privat:**
 - `_owner: str`
Menyimpan nama pemilik akun (tipe data string).
 - `_balance: int`
Menyimpan saldo akun (tipe data integer).
- **Metode Publik:**
 - `__init__(owner: str, balance: int)`
Konstruktor yang digunakan untuk menginisialisasi objek dengan parameter `owner` dan `balance`.
 - `deposit(amount: int): None`
Metode untuk menambahkan saldo ke akun. Jika nilai `amount` lebih dari 0, maka nilai tersebut ditambahkan ke `_balance`.
 - `withdraw(amount: int): None`
Metode untuk menarik saldo. Melakukan validasi agar jumlah penarikan tidak melebihi saldo yang tersedia.
 - `get_balance(): int`
Metode untuk mengambil nilai saldo akun saat ini.

Diagram di atas memberikan gambaran mengenai struktur kelas **BankAccount** serta cara kerja enkapsulasi dalam mengatur atribut dan metode di dalam kelas. Kemudian untuk kode praktikum dalam python dapat dilihat sebagai berikut:

```

01: class BankAccount:
02:     def __init__(self, owner, balance):
03:         # Atribut dengan double underscore (_) dianggap "private" di
Python
04:         self.__owner = owner
05:         self.__balance = balance
06:
  
```

```

07:     def deposit(self, amount):
08:         """Method untuk menambahkan saldo."""
09:         if amount > 0:
10:             self.__balance += amount
11:             print(f"{amount} telah ditambahkan ke akun {self.__owner}.")
12:         else:
13:             print("Jumlah deposit harus lebih dari 0.")
14:
15:     def withdraw(self, amount):
16:         """Method untuk menarik saldo."""
17:         if amount <= self.__balance:
18:             self.__balance -= amount
19:             print(f"{amount} telah ditarik dari akun {self.__owner}.")
20:         else:
21:             print("Saldo tidak mencukupi.")
22:
23:     def get_balance(self):
24:         """Method untuk mendapatkan informasi saldo terkini."""
25:         return self.__balance
26:
27: # Contoh penggunaan
28: if __name__ == "__main__":
29:     # Membuat objek BankAccount dengan owner="Alice" dan balance awal
1000
30:     alice_account = BankAccount(owner="Alice", balance=1000)
31:
32:     # Deposit uang
33:     alice_account.deposit(500)      # Berhasil
34:     alice_account.deposit(-100)     # Gagal (validasi)
35:
36:     # Withdraw uang
37:     alice_account.withdraw(300)    # Berhasil
38:     alice_account.withdraw(2000)   # Gagal (saldo tidak cukup)
39:
40:     # Mendapatkan saldo
41:     current_balance = alice_account.get_balance()
42:     print(f"Saldo terakhir di akun {alice_account._BankAccount__owner}: {current_balance}")
43:
44:         # Mencoba mengakses atribut 'private' langsung (tidak direkomendasikan)
45:         # alice_account.__balance # Akan error
46:         # Karena Python "mangling" nama atribut __balance menjadi _BankAccount__balance
47:         # Ini adalah salah satu mekanisme enkapsulasi sederhana di Python.
48:

```

Penjelasan

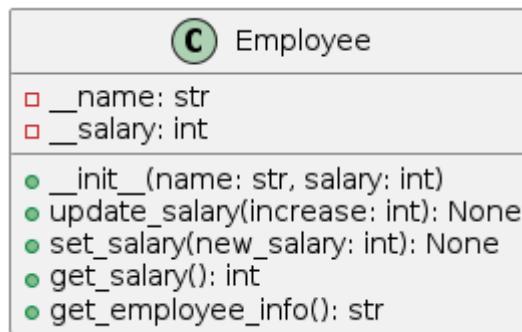
- Atribut Private:** Di Python, penulisan `__nama_atribut` membuat nama atribut tersebut di-“mangled” sehingga tidak dapat diakses langsung dari luar kelas dengan nama aslinya. Contoh: `__balance` menjadi `_BankAccount__balance`. Ini bukan perlindungan mutlak, tetapi menjadi sinyal kuat bahwa atribut tersebut tidak untuk diakses sembarangan.
- Metode Akses (Get/Set):** Konsep “getter” atau “setter” diterapkan untuk mengontrol validasi dan logika ketika atribut diubah. Di contoh ini, method `deposit()` dan `withdraw()` melindungi `__balance` dari perubahan langsung dengan menambahkan sejumlah kondisi (validasi).
- Keuntungan Enkapsulasi:**

-
- Memudahkan perawatan kode (maintenance) karena data tidak diubah sembarangan.
 - Mempermudah menambahkan validasi/pengecekan keamanan (security check).
 - Mengontrol akses terhadap data, sehingga lebih mudah dikelola.

Meski Python tidak memiliki sistem enkapsulasi “private”/“protected” secara ketat seperti beberapa bahasa pemrograman lain, penulisan underscore ganda dan pendekatan getter/setter tetap menjadi praktik umum untuk menegaskan bahwa atribut atau metode tersebut bersifat internal.

Praktikum 02: Enkapsulasi pada kelas Employee

Berikut ini adalah contoh lain tentang enkapsulasi dalam Python menggunakan kelas Employee yang menyembunyikan data karyawan, seperti nama dan gaji, dari akses langsung di luar kelas. Pada contoh ini, kita menyediakan method khusus (getter dan setter) untuk mengakses dan memodifikasi data privat. UML kelass diagram untuk Employee Class.



- **Atribut Privat (-):**
 - `__name: str` → Menyimpan nama karyawan.
 - `__salary: int` → Menyimpan gaji karyawan.
- **Metode Publik (+):**
 - `__init__(name, salary)` → Konstruktor untuk inisialisasi nama dan gaji.
 - `update_salary(increase)` → Menambahkan gaji jika nilai kenaikan valid.
 - `set_salary(new_salary)` → Mengatur ulang gaji dengan validasi.
 - `get_salary()` → Mengembalikan nilai gaji saat ini.
 - `get_employee_info()` → Menampilkan info lengkap tentang karyawan.

Kode Python:

```

01: class Employee:
02:     def __init__(self, name, salary):
03:         # Atribut privat dengan double underscore
04:         self.__name = name
05:         self.__salary = salary
06:
07:     def update_salary(self, increase):
08:         """Method untuk menaikkan gaji dengan validasi."""
09:         if increase > 0:
10:             self.__salary += increase
11:             print(f"Gaji telah dinaikkan sebesar {increase}.")
12:         else:
13:             print("Nilai kenaikan harus lebih dari 0.")
14:
15:     def set_salary(self, new_salary):
16:         """Method untuk mengubah gaji dengan validasi."""
17:         if new_salary >= 0:
18:             self.__salary = new_salary
19:             print(f"Gaji diatur ulang menjadi {new_salary}.")
20:         else:
21:             print("Gaji tidak dapat bernilai negatif.")
22:
23:     def get_salary(self):
24:         """Method untuk mendapatkan informasi gaji."""
25:         return self.__salary
26:
27:     def get_employee_info(self):
  
```

```

28:         """Method untuk menampilkan informasi karyawan secara
menyeluruh."""
29:         return f"Employee: {self.__name}, Gaji: {self.__salary}"
30:
31: # Contoh penggunaan
32: if __name__ == "__main__":
33:     # Membuat objek Employee dengan nama "John Doe" dan gaji awal 50000
34:     employee1 = Employee("John Doe", 50000)
35:
36:     # Tampilkan informasi karyawan
37:     print(employee1.get_employee_info())
38:
39:     # Update gaji dengan menaikkan sebesar 5000
40:     employee1.update_salary(5000)
41:     print(f"Gaji setelah kenaikan: {employee1.get_salary()}")
42:
43:     # Atur ulang gaji dengan nilai baru
44:     employee1.set_salary(60000)
45:     print(f"Informasi terbaru: {employee1.get_employee_info()}")
46:
47:     # Mencoba mengakses atribut privat secara langsung (tidak
direkomendasikan)
48:     # Contoh: print(employee1.__salary) --> Ini akan menimbulkan error
49:

```

Penjelasan

1. Atribut Privat

- Atribut `__name` dan `__salary` dimulai dengan dua garis bawah (double underscore), yang menandakan bahwa atribut tersebut bersifat privat.
- Python melakukan name mangling sehingga atribut ini tidak dapat diakses secara langsung dari luar kelas, melainkan harus menggunakan method yang disediakan.

2. Method untuk Memodifikasi Data

- `update_salary()`: Menambahkan nilai kenaikan ke gaji saat ini dengan validasi bahwa nilai kenaikan harus positif.
- `set_salary()`: Mengatur nilai gaji secara langsung asalkan nilai tersebut tidak negatif.

3. Method untuk Mengakses Data

- `get_salary()`: Mengembalikan nilai gaji saat ini.
- `get_employee_info()`: Mengembalikan informasi lengkap tentang karyawan termasuk nama dan gaji.

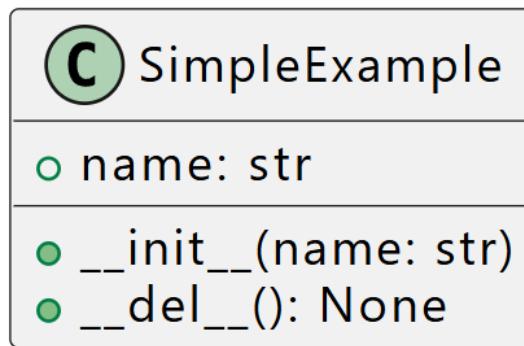
4. Keuntungan Enkapsulasi

- Data sensitif seperti gaji disembunyikan sehingga hanya dapat diubah melalui method yang telah ditentukan.
- Validasi data dilakukan di dalam method untuk menjaga agar nilai yang tidak valid tidak masuk ke atribut.
- Menjaga integritas data dan memudahkan perawatan kode.

Dengan pendekatan ini, kita memastikan bahwa data dalam objek `Employee` aman dari modifikasi secara langsung dari luar kelas dan hanya dapat diubah melalui operasi yang telah diatur sesuai logika bisnis.

Praktikum 03: Konstruktor dan Destruktor Sederhana

Berikut adalah contoh program sederhana yang mendemonstrasikan cara kerja konstruktor dan destruktur dalam Python:



- **Atribut:**
 - name: str → Atribut publik yang disimpan saat objek dibuat.
- **Method:**
 - __init__(name: str) → Konstruktor yang dipanggil saat objek diinisialisasi.
 - __del__() → Destruktor yang dipanggil saat objek dihapus (baik secara eksplisit dengan del atau otomatis oleh garbage collector).

Kode Program

```

01: class SimpleExample:
02:     def __init__(self, name):
03:         """
04:             Konstruktor: Dipanggil saat objek dibuat.
05:             Menyimpan nilai 'name' dan mencetak pesan pembuatan objek.
06:         """
07:         self.name = name
08:         print(f"Konstruktor: Objek '{self.name}' telah dibuat.")

09:
10:     def __del__(self):
11:         """
12:             Destruktor: Dipanggil saat objek dihapus.
13:             Mencetak pesan bahwa objek sedang dihapus.
14:         """
15:         print(f"Destruktor: Objek '{self.name}' sedang dihapus.")

16:
17:
18: def main():
19:     print("Program dimulai.\n")
20:
21:     # Membuat objek SimpleExample
22:     obj = SimpleExample("Demo")
23:     print("Program sedang berjalan... \n")
24:
25:     # Menghapus objek secara eksplisit
26:     del obj
27:     print("Objek telah dihapus secara eksplisit.\n")
28:
29:     print("Program selesai.")
30:
31: if __name__ == "__main__":
32:     main()

```

1. Konstruktor (`__init__`):

Saat objek dibuat dengan `SimpleExample("Demo")`, Python secara otomatis memanggil metode `__init__`, yang mencetak pesan bahwa objek telah dibuat dan menyimpan atribut `name`.

2. Destruktor (`__del__`):

Ketika perintah `del` obj dipanggil, objek dihapus secara eksplisit (meskipun pada kenyataannya Python akan menghapus objek ketika tidak ada lagi referensi), sehingga metode `__del__` terpanggil dan mencetak pesan bahwa objek sedang dihapus.

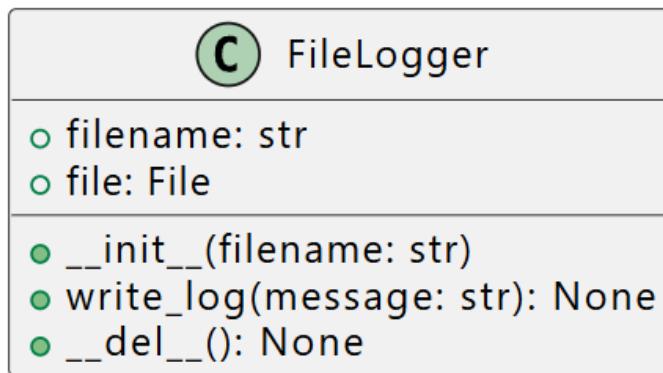
3. Alur Program:

Program mulai dengan pembuatan objek, melakukan beberapa operasi (di sini hanya mencetak beberapa pesan), lalu menghapus objek dan menampilkan pesan bahwa objek telah dihapus.

Program ini memberikan gambaran yang sederhana mengenai bagaimana konstruktur dan destruktur bekerja di Python untuk mengelola siklus hidup objek.

Praktikum 04: Konstruktor dan Destruktor program FileLogger

Berikut adalah contoh lain yang lebih nyata, yakni simulasi pengelolaan koneksi jaringan. Dalam contoh ini, kelas **NetworkConnection** membuat "koneksi" ke sebuah alamat host dan port ketika objek dibuat (menggunakan konstruktor) dan menutup koneksi tersebut ketika objek dihapus (menggunakan destruktur). Selain itu, terdapat metode untuk mengirim dan menerima data melalui koneksi tersebut.



- ✓ **Atribut:**
 - filename: str → Nama file tempat log ditulis.
 - file: File → Objek file yang dibuka dalam mode append ("a"), digunakan untuk menulis log.
- ✓ **Method:**
 - __init__(filename: str) → Konstruktor untuk membuka file log saat objek dibuat.
 - write_log(message: str) → Menulis pesan log ke file dan flush ke disk.
 - __del__() → Destruktor untuk menutup file saat objek dihapus.

Kode Program:

```

01: class FileLogger:
02:     def __init__(self, filename):
03:         """
04:             Konstruktor: Membuka file log untuk menulis pesan.
05:             Parameter:
06:                 - filename: Nama file tempat pesan log akan ditulis.
07:         """
08:         self.filename = filename
09:         try:
10:             self.file = open(filename, "a")    # Membuka file dalam mode
append
11:             print(f"File '{filename}' berhasil dibuka untuk logging.")
12:         except Exception as e:
13:             print(f"Gagal membuka file '{filename}': {e}")
14:
15:     def write_log(self, message):
16:         """
17:             Menulis pesan log ke dalam file.
18:             Parameter:
19:                 - message: Pesan yang akan ditulis ke file.
20:         """
21:             self.file.write(message + "\n")
22:             self.file.flush()  # Memastikan pesan langsung ditulis ke disk
23:             print(f"pesan log: '{message}' telah ditulis.")
24:
25:     def __del__(self):
26:         """
  
```

```

27:         Destruktor: Menutup file log ketika objek dihapus.
28:         """
29:         if hasattr(self, "file") and not self.file.closed:
30:             self.file.close()
31:             print(f"File '{self.filename}' telah ditutup.")
32:
33:
34: # Contoh penggunaan dalam skenario nyata aplikasi
35: if __name__ == "__main__":
36:     # Membuat objek logger untuk file "application.log"
37:     logger = FileLogger("application.log")
38:
39:     # Menulis beberapa pesan log selama operasi aplikasi
40:     logger.write_log("Aplikasi dimulai.")
41:     logger.write_log("Melakukan operasi A...")
42:     logger.write_log("Operasi A selesai.")
43:     logger.write_log("Aplikasi akan segera selesai.")
44:
45:     # Menghapus objek logger secara eksplisit
46:     del logger
47:
48:     # Jika objek tidak dihapus secara eksplisit, destruktur akan dipanggil
49:     # ketika program berakhir dan garbage collection membersihkan objek
tersebut.
50:

```

1. **Konstruktor (`__init__`):**

- Saat objek **NetworkConnection** dibuat, konstruktor akan dijalankan untuk membuka koneksi ke alamat host dan port yang diberikan.
- Atribut `self.connected` diset ke True untuk menandakan koneksi aktif, dan delay dengan `time.sleep(1)` digunakan untuk mensimulasikan proses inisialisasi koneksi.

2. **Metode `send_data` dan `receive_data`:**

- **`send_data(data)`:** Metode ini menampilkan pesan pengiriman data, mensimulasikan delay singkat, dan mengonfirmasi bahwa data telah dikirim selama koneksi dalam keadaan aktif.
- **`receive_data()`:** Metode ini mensimulasikan penerimaan data dengan delay, mengembalikan pesan yang diterima, dan menampilkan hasilnya ke layar.

3. **Destruktor (`__del__`):**

- Saat objek dihapus, destruktur akan dijalankan untuk menutup koneksi jika koneksi masih aktif.
- Proses penutupan juga disimulasikan dengan delay, dan status koneksi diubah menjadi False.

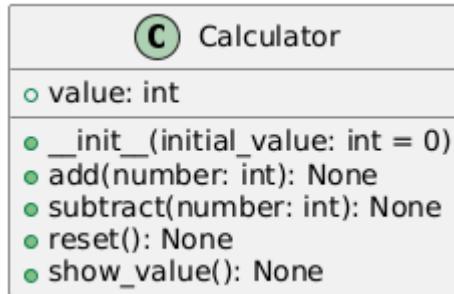
4. **Alur Program di Fungsi `main()`:**

- Program dimulai dengan membuat objek **NetworkConnection**.
- Setelah itu, program mengirim dan menerima data melalui koneksi yang telah dibuat.
- Objek koneksi kemudian dihapus secara eksplisit dengan `del conn`, sehingga destruktur segera dipanggil untuk menutup koneksi.
- Setelah menunggu sejenak, program selesai.

Contoh ini menggambarkan bagaimana konstruktor dan destruktur bekerja dalam konteks nyata, yaitu dalam pengelolaan sumber daya (resource) berupa koneksi jaringan. Teknik ini penting untuk memastikan bahwa resource dilepaskan dengan benar, menghindari kebocoran resource, dan menjaga integritas aplikasi.

Praktikum 05: Properti "self" pada kelas

Berikut adalah contoh kode Python yang mendemonstrasikan peran dari **self** dalam metode sebuah kelas. Kode ini menjelaskan bagaimana **self** digunakan untuk merujuk ke atribut dan metode instance, sehingga setiap objek (instance) memiliki data dan perilaku yang independen. Diagram UML Kelas Calculator sebagai berikut:



- **Atribut:**
 - value: int → Nilai saat ini yang disimpan dalam kalkulator (bersifat publik dalam kode Python).
- **Method:**
 - __init__(initial_value: int = 0) → Konstruktor untuk inisialisasi nilai awal kalkulator.
 - add(number: int) → Menambahkan angka ke nilai saat ini.
 - subtract(number: int) → Mengurangi angka dari nilai saat ini.
 - reset() → Mengatur nilai kalkulator kembali ke nol.
 - show_value() → Menampilkan nilai saat ini.

Kode Program:

```

01: class Calculator:
02:     def __init__(self, initial_value=0):
03:         """
04:             Konstruktor kelas Calculator.
05:             - self: Mengacu pada instance yang dibuat.
06:             - initial_value: Nilai awal dari kalkulator.
07:         """
08:         self.value = initial_value
09:         print(f"Kalkulator diinisialisasi dengan nilai: {self.value}")
10:
11:     def add(self, number):
12:         """
13:             Menambahkan 'number' ke nilai yang tersimpan di objek.
14:             - self.value: Menunjukkan nilai saat ini yang disimpan di objek tersebut.
15:             - number: Nilai yang akan ditambahkan.
16:         """
17:         self.value += number
18:         print(f"Setelah penambahan {number}, nilai sekarang adalah: {self.value}")
19:
20:     def subtract(self, number):
21:         """
22:             Mengurangi 'number' dari nilai yang tersimpan.
23:             - self.value: Nilai saat ini dalam objek.
24:             - number: Nilai yang akan dikurangkan.
25:         """
26:         self.value -= number
  
```

```

27:         print(f"Setelah pengurangan {number}, nilai sekarang adalah:
{self.value}")
28:
29:     def reset(self):
30:         """
31:             Mengatur ulang nilai kalkulator ke 0.
32:         """
33:         self.value = 0
34:         print("Nilai telah direset ke 0.")
35:
36:     def show_value(self):
37:         """
38:             Menampilkan nilai saat ini dari kalkulator.
39:         """
40:         print(f"Nilai saat ini adalah: {self.value}")
41:
42: # Contoh penggunaan untuk memahami peran 'self'
43: def main():
44:     # Membuat objek Calculator dengan nilai awal 10
45:     calc1 = Calculator(initial_value=10)
46:
47:     # Menggunakan metode dari objek calc1
48:     calc1.add(5)          # Menambah 5 ke nilai calc1
49:     calc1.subtract(3)     # Mengurangi 3 dari nilai calc1
50:     calc1.show_value()    # Menampilkan nilai calc1
51:
52:     # Membuat objek Calculator lainnya dengan nilai awal default (0)
53:     calc2 = Calculator()
54:     calc2.add(20)         # Menambah 20 ke nilai calc2
55:     calc2.show_value()    # Menampilkan nilai calc2
56:
57:     # Penjelasan peran self:
58:     # 'self' memungkinkan setiap instance (calc1, calc2) memiliki atribut
59:     # 'value' masing-masing.
60:     # Perubahan yang dilakukan pada calc1 tidak akan mempengaruhi calc2,
61:     # karena masing-masing
62:     # mengacu pada self yang berbeda (instance yang berbeda).
63: if __name__ == "__main__":
64:     main()

```

Penjelasan Kode

1. Konstruktor (`__init__`)

- Parameter **self** merujuk pada instance yang dibuat.
- Atribut **self.value** diinisialisasi dengan nilai awal yang diberikan.
- Setiap kali objek baru dibuat, konstruktor ini dipanggil secara otomatis.

2. Method `add()`, `subtract()`, `reset()`, dan `show_value()`

- Semua method memiliki parameter pertama **self** yang memungkinkan mereka mengakses dan mengubah atribut instance.
- Misalnya, ketika `calc1.add(5)` dipanggil, method `add()` menggunakan **self** untuk menambah 5 ke `calc1.value`.

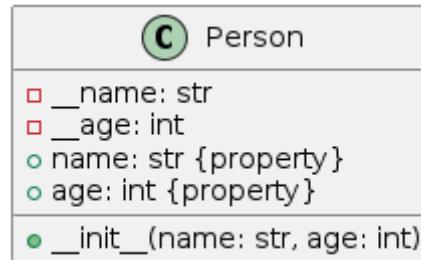
3. Penggunaan Banyak Instance

- Pada bagian `main()`, dua objek `calc1` dan `calc2` dibuat secara terpisah.
- Perubahan yang terjadi pada `calc1` tidak berpengaruh pada `calc2`, karena masing-masing memiliki atribut **self.value** mereka sendiri.

Kode ini memberikan gambaran jelas tentang bagaimana **self** bekerja untuk mengakses dan memanipulasi data dalam objek Python.

Praktikum 06: Menggunakan getter, setter, dan dekorator @property untuk mengelola akses atribut

Di bawah ini adalah contoh kode Python yang menjelaskan cara menggunakan **getter**, **setter**, dan dekorator **@property** untuk mengelola akses ke atribut yang bersifat privat. Contoh ini menggunakan kelas Person yang memiliki atribut privat `__name` dan `__age`. Getter dan setter digunakan untuk mengakses dan memodifikasi atribut tersebut dengan validasi yang sesuai. UML kelas diagram Person.



- **Atribut Privat (-):**
 - `__name: str` – disimpan secara internal, tidak bisa diakses langsung dari luar kelas.
 - `__age: int` – juga bersifat privat.
- **Konstruktor (+ `__init__(name, age)`):**
 - Digunakan untuk menginisialisasi objek dengan nilai awal untuk name dan age.
- **Property Methods ({property}):**
 - name dan age menggunakan dekorator `@property`, sehingga dapat digunakan seperti atribut biasa tapi dengan pengelolaan akses (getter/setter) di balik layar.

Kode Program:

```

01: class Person:
02:     def __init__(self, name, age):
03:         """
04:             Konstruktor untuk menginisialisasi objek Person dengan nama dan
05:             umur.
06:             Atribut privat (dengan double underscore) menyimpan data internal.
07:             """
08:         self.__name = name
09:         self.__age = age
10:
11:     @property
12:     def name(self):
13:         """
14:             Getter untuk atribut name.
15:             Mengembalikan nilai dari __name.
16:             """
17:         return self.__name
18:
19:     @name.setter
20:     def name(self, value):
21:         """
22:             Setter untuk atribut name.
23:             Memeriksa apakah nilai tidak kosong sebelum mengubah nilai
24:             name.
25:             """
26:             if not value:
27:                 print("Nama tidak boleh kosong.")
28:             else:
29:
30:
31:
32:
33:
34:
35:
36:
37:
38:
39:
40:
41:
42:
43:
44:
45:
46:
47:
48:
49:
50:
51:
52:
53:
54:
55:
56:
57:
58:
59:
60:
61:
62:
63:
64:
65:
66:
67:
68:
69:
70:
71:
72:
73:
74:
75:
76:
77:
78:
79:
80:
81:
82:
83:
84:
85:
86:
87:
88:
89:
90:
91:
92:
93:
94:
95:
96:
97:
98:
99:
100:
101:
102:
103:
104:
105:
106:
107:
108:
109:
110:
111:
112:
113:
114:
115:
116:
117:
118:
119:
120:
121:
122:
123:
124:
125:
126:
127:
128:
129:
130:
131:
132:
133:
134:
135:
136:
137:
138:
139:
140:
141:
142:
143:
144:
145:
146:
147:
148:
149:
150:
151:
152:
153:
154:
155:
156:
157:
158:
159:
160:
161:
162:
163:
164:
165:
166:
167:
168:
169:
170:
171:
172:
173:
174:
175:
176:
177:
178:
179:
180:
181:
182:
183:
184:
185:
186:
187:
188:
189:
190:
191:
192:
193:
194:
195:
196:
197:
198:
199:
200:
201:
202:
203:
204:
205:
206:
207:
208:
209:
210:
211:
212:
213:
214:
215:
216:
217:
218:
219:
220:
221:
222:
223:
224:
225:
226:
227:
228:
229:
230:
231:
232:
233:
234:
235:
236:
237:
238:
239:
240:
241:
242:
243:
244:
245:
246:
247:
248:
249:
250:
251:
252:
253:
254:
255:
256:
257:
258:
259:
259:
260:
261:
262:
263:
264:
265:
266:
267:
268:
269:
269:
270:
271:
272:
273:
274:
275:
276:
277:
278:
279:
279:
280:
281:
282:
283:
284:
285:
286:
287:
288:
289:
289:
290:
291:
292:
293:
294:
295:
296:
297:
298:
299:
299:
300:
301:
302:
303:
304:
305:
306:
307:
308:
309:
309:
310:
311:
312:
313:
314:
315:
316:
317:
318:
319:
319:
320:
321:
322:
323:
324:
325:
326:
327:
328:
329:
329:
330:
331:
332:
333:
334:
335:
336:
337:
337:
338:
339:
339:
340:
341:
342:
343:
344:
345:
345:
346:
347:
347:
348:
349:
349:
350:
351:
352:
353:
354:
355:
355:
356:
357:
357:
358:
359:
359:
360:
361:
362:
363:
364:
364:
365:
366:
366:
367:
367:
368:
368:
369:
369:
370:
371:
371:
372:
372:
373:
373:
374:
374:
375:
375:
376:
376:
377:
377:
378:
378:
379:
379:
380:
380:
381:
381:
382:
382:
383:
383:
384:
384:
385:
385:
386:
386:
387:
387:
388:
388:
389:
389:
390:
390:
391:
391:
392:
392:
393:
393:
394:
394:
395:
395:
396:
396:
397:
397:
398:
398:
399:
399:
400:
400:
401:
401:
402:
402:
403:
403:
404:
404:
405:
405:
406:
406:
407:
407:
408:
408:
409:
409:
410:
410:
411:
411:
412:
412:
413:
413:
414:
414:
415:
415:
416:
416:
417:
417:
418:
418:
419:
419:
420:
420:
421:
421:
422:
422:
423:
423:
424:
424:
425:
425:
426:
426:
427:
427:
428:
428:
429:
429:
430:
430:
431:
431:
432:
432:
433:
433:
434:
434:
435:
435:
436:
436:
437:
437:
438:
438:
439:
439:
440:
440:
441:
441:
442:
442:
443:
443:
444:
444:
445:
445:
446:
446:
447:
447:
448:
448:
449:
449:
450:
450:
451:
451:
452:
452:
453:
453:
454:
454:
455:
455:
456:
456:
457:
457:
458:
458:
459:
459:
460:
460:
461:
461:
462:
462:
463:
463:
464:
464:
465:
465:
466:
466:
467:
467:
468:
468:
469:
469:
470:
470:
471:
471:
472:
472:
473:
473:
474:
474:
475:
475:
476:
476:
477:
477:
478:
478:
479:
479:
480:
480:
481:
481:
482:
482:
483:
483:
484:
484:
485:
485:
486:
486:
487:
487:
488:
488:
489:
489:
490:
490:
491:
491:
492:
492:
493:
493:
494:
494:
495:
495:
496:
496:
497:
497:
498:
498:
499:
499:
500:
500:
501:
501:
502:
502:
503:
503:
504:
504:
505:
505:
506:
506:
507:
507:
508:
508:
509:
509:
510:
510:
511:
511:
512:
512:
513:
513:
514:
514:
515:
515:
516:
516:
517:
517:
518:
518:
519:
519:
520:
520:
521:
521:
522:
522:
523:
523:
524:
524:
525:
525:
526:
526:
527:
527:
528:
528:
529:
529:
530:
530:
531:
531:
532:
532:
533:
533:
534:
534:
535:
535:
536:
536:
537:
537:
538:
538:
539:
539:
540:
540:
541:
541:
542:
542:
543:
543:
544:
544:
545:
545:
546:
546:
547:
547:
548:
548:
549:
549:
550:
550:
551:
551:
552:
552:
553:
553:
554:
554:
555:
555:
556:
556:
557:
557:
558:
558:
559:
559:
560:
560:
561:
561:
562:
562:
563:
563:
564:
564:
565:
565:
566:
566:
567:
567:
568:
568:
569:
569:
570:
570:
571:
571:
572:
572:
573:
573:
574:
574:
575:
575:
576:
576:
577:
577:
578:
578:
579:
579:
580:
580:
581:
581:
582:
582:
583:
583:
584:
584:
585:
585:
586:
586:
587:
587:
588:
588:
589:
589:
590:
590:
591:
591:
592:
592:
593:
593:
594:
594:
595:
595:
596:
596:
597:
597:
598:
598:
599:
599:
600:
600:
601:
601:
602:
602:
603:
603:
604:
604:
605:
605:
606:
606:
607:
607:
608:
608:
609:
609:
610:
610:
611:
611:
612:
612:
613:
613:
614:
614:
615:
615:
616:
616:
617:
617:
618:
618:
619:
619:
620:
620:
621:
621:
622:
622:
623:
623:
624:
624:
625:
625:
626:
626:
627:
627:
628:
628:
629:
629:
630:
630:
631:
631:
632:
632:
633:
633:
634:
634:
635:
635:
636:
636:
637:
637:
638:
638:
639:
639:
640:
640:
641:
641:
642:
642:
643:
643:
644:
644:
645:
645:
646:
646:
647:
647:
648:
648:
649:
649:
650:
650:
651:
651:
652:
652:
653:
653:
654:
654:
655:
655:
656:
656:
657:
657:
658:
658:
659:
659:
660:
660:
661:
661:
662:
662:
663:
663:
664:
664:
665:
665:
666:
666:
667:
667:
668:
668:
669:
669:
670:
670:
671:
671:
672:
672:
673:
673:
674:
674:
675:
675:
676:
676:
677:
677:
678:
678:
679:
679:
680:
680:
681:
681:
682:
682:
683:
683:
684:
684:
685:
685:
686:
686:
687:
687:
688:
688:
689:
689:
690:
690:
691:
691:
692:
692:
693:
693:
694:
694:
695:
695:
696:
696:
697:
697:
698:
698:
699:
699:
700:
700:
701:
701:
702:
702:
703:
703:
704:
704:
705:
705:
706:
706:
707:
707:
708:
708:
709:
709:
710:
710:
711:
711:
712:
712:
713:
713:
714:
714:
715:
715:
716:
716:
717:
717:
718:
718:
719:
719:
720:
720:
721:
721:
722:
722:
723:
723:
724:
724:
725:
725:
726:
726:
727:
727:
728:
728:
729:
729:
730:
730:
731:
731:
732:
732:
733:
733:
734:
734:
735:
735:
736:
736:
737:
737:
738:
738:
739:
739:
740:
740:
741:
741:
742:
742:
743:
743:
744:
744:
745:
745:
746:
746:
747:
747:
748:
748:
749:
749:
750:
750:
751:
751:
752:
752:
753:
753:
754:
754:
755:
755:
756:
756:
757:
757:
758:
758:
759:
759:
760:
760:
761:
761:
762:
762:
763:
763:
764:
764:
765:
765:
766:
766:
767:
767:
768:
768:
769:
769:
770:
770:
771:
771:
772:
772:
773:
773:
774:
774:
775:
775:
776:
776:
777:
777:
778:
778:
779:
779:
780:
780:
781:
781:
782:
782:
783:
783:
784:
784:
785:
785:
786:
786:
787:
787:
788:
788:
789:
789:
790:
790:
791:
791:
792:
792:
793:
793:
794:
794:
795:
795:
796:
796:
797:
797:
798:
798:
799:
799:
800:
800:
801:
801:
802:
802:
803:
803:
804:
804:
805:
805:
806:
806:
807:
807:
808:
808:
809:
809:
810:
810:
811:
811:
812:
812:
813:
813:
814:
814:
815:
815:
816:
816:
817:
817:
818:
818:
819:
819:
820:
820:
821:
821:
822:
822:
823:
823:
824:
824:
825:
825:
826:
826:
827:
827:
828:
828:
829:
829:
830:
830:
831:
831:
832:
832:
833:
833:
834:
834:
835:
835:
836:
836:
837:
837:
838:
838:
839:
839:
840:
840:
841:
841:
842:
842:
843:
843:
844:
844:
845:
845:
846:
846:
847:
847:
848:
848:
849:
849:
850:
850:
851:
851:
852:
852:
853:
853:
854:
854:
855:
855:
856:
856:
857:
857:
858:
858:
859:
859:
860:
860:
861:
861:
862:
862:
863:
863:
864:
864:
865:
865:
866:
866:
867:
867:
868:
868:
869:
869:
870:
870:
871:
871:
872:
872:
873:
873:
874:
874:
875:
875:
876:
876:
877:
877:
878:
878:
879:
879:
880:
880:
881:
881:
882:
882:
883:
883:
884:
884:
885:
885:
886:
886:
887:
887:
888:
888:
889:
889:
890:
890:
891:
891:
892:
892:
893:
893:
894:
894:
895:
895:
896:
896:
897:
897:
898:
898:
899:
899:
900:
900:
901:
901:
902:
902:
903:
903:
904:
904:
905:
905:
906:
906:
907:
907:
908:
908:
909:
909:
910:
910:
911:
911:
912:
912:
913:
913:
914:
914:
915:
915:
916:
916:
917:
917:
918:
918:
919:
919:
920:
920:
921:
921:
922:
922:
923:
923:
924:
924:
925:
925:
926:
926:
927:
927:
928:
928:
929:
929:
930:
930:
931:
931:
932:
932:
933:
933:
934:
934:
935:
935:
936:
936:
937:
937:
938:
938:
939:
939:
940:
940:
941:
941:
942:
942:
943:
943:
944:
944:
945:
945:
946:
946:
947:
947:
948:
948:
949:
949:
950:
950:
951:
951:
952:
952:
953:
953:
954:
954:
955:
955:
956:
956:
957:
957:
958:
958:
959:
959:
960:
960:
961:
961:
962:
962:
963:
963:
964:
964:
965:
965:
966:
966:
967:
967:
968:
968:
969:
969:
970:
970:
971:
971:
972:
972:
973:
973:
974:
974:
975:
975:
976:
976:
977:
977:
978:
978:
979:
979:
980:
980:
981:
981:
982:
982:
983:
983:
984:
984:
985:
985:
986:
986:
987:
987:
988:
988:
989:
989:
990:
990:
991:
991:
992:
992:
993:
993:
994:
994:
995:
995:
996:
996:
997:
997:
998:
998:
999:
999:
1000:
1000:
1001:
1001:
1002:
1002:
1003:
1003:
1004:
1004:
1005:
1005:
1006:
1006:
1007:
1007:
1008:
1008:
1009:
1009:
1010:
1010:
1011:
1011:
1012:
1012:
1013:
1013:
1014:
1014:
1015:
1015:
1016:
1016:
1017:
1017:
1018:
1018:
1019:
1019:
1020:
1020:
1021:
1021:
1022:
1022:
1023:
1023:
1024:
1024:
1025:
1025:
1026:
1026:
1027:
1027:
1028:
1028:
1029:
1029:
1030:
1030:
1031:
1031:
1032:
1032:
1033:
1033:
1034:
1034:
1035:
1035:
1036:
1036:
1037:
1037:
1038:
1038:
1039:
1039:
1040:
1040:
1041:
1041:
1042:
1042:
1043:
1043:
1044:
1044:
1045:
1045:
1046:
1046:
1047:
1047:
1048:
1048:
1049:
1049:
1050:
1050:
1051:
1051:
1052:
1052:
1053:
1053:
1054:
1054:
1055:
1055:
1056:
1056:
1057:
1057:
1058:
1058:
1059:
1059:
1060:
1060:
1061:
1061:
1062:
1062:
1063:
1063:
1064:
1064:
1065:
1065:
1066:
1066:
1067:
1067:
1068:
1068:
1069:
1069:
1070:
1070:
1071:
1071:
1072:
1072:
1073:
1073:
1074:
1074:
1075:
1075:
1076:
1076:
1077:
1077:
1078:
1078:
1079:
1079:
1080:
1080:
1081:
1081:
1082:
1082:
1083:
1083:
1084:
1084:
1085:
1085:
1086:
1086:
1087:
1087:
1088:
1088:
1089:
1089:
1090:
1090:
1091:
1091:
1092:
1092:
1093:
1093:
1094:
1094:
1095:
1095:
1096:
1096:
1097:
1097:
1098:
1098:
1099:
1099:
1100:
1100:
1101:
1101:
1102:
1102:
1103:
1103:
1104:
1104:
1105:
1105:
1106:
1106:
1107:
1107:
1108:
1108:
1109:
1109:
1110:
1110:
1111:
1111:
1112:
1112:
1113:
1113:
1114:
1114:
1115:
1115:
1116:
1116:
1117:
1117:
1118:
1118:
1119:
1119:
1120:
1120:
1121:
1121:
1122:
1122:
1123:
1123:
1124:
1124:
1125:
1125:
1126:
1126:
1127:
1127:
1128:
1128:
1129:
1129:
1130:
1130:
1131:
1131:
1132:
1132:
1133:
1133:
1134:
1134:
1135:
1135:
1136:
1136:
1137:
1137:
1138:
1138:
1139:
1139:
1140:
1140:
1141:
1141:
1142:
1142:
1143:
1143:
1144:
1144:
1145:
1145:
1146:
1146:
1147:
1147:
1148:
1148:
1149:
1149:
1150:
1150:
1151:
1151:
1152:
1152:
1153:
1153:
1154:
1154:
1155:
1155:
1156:
1156:
1157:
1157:
1158:
1158:
1159:
1159:
1160:
1160:
1161:
1161:
1162:
1162:
1163:
1163:
1164:
1164:
1165:
1165:
1166:
1166:
1167:
1167:
1168:
1168:
1169:
1169:
1170:
1170:
1171:
1171:
1172:
1172:
1173:
1173:
1174:
1174:
1175:
1175:
1176:
1176:
1177:
1177:
1178:
1178:
1179:
1179:
1180:
1180:
1181:
1181:
1182:
1182:
1183:
1183:
1184:
1184:
1185:
1185:
1186:
1186:
1187:
1187:
1188:
1188:
1189:
1189:
1190:
1190:
1191:
1191:
1192:
1192:
1193:
1193:
1194:
1194:
1195:
1195:
1196:
1196:
1197:
1197:
1198:
1198:
1199:
1199:
1200:
1200:
1201:
1201:
1202:
1202:
1203:
1203:
1204:
1204:
1205:
1205:
1206:
1206:
1207:
1207:
1208:
1208:
1209:
1209:
1210:
1210:
1211:
1211:
1212:
1212:
1213:
1213:
1214:
1214:
1215:
1215:
1216:
1216:
1217:
1217:
1218:
1218:
1219:
1219:
1220:
1220:
1221:
1221:
1222:
1222:
1223:
1223:
1224:
1224:
1225:
1225:
1226:
1226:
1227:
1227:
1228:
1228:
1229:
1229:
1230:
1230:
1231:
1231:
1232:
1232:
1233:
1233:
1234:
1234:
1235:
1235:
1236:
1236:
1237:
1237:
1238:
1238:
1239:
1239:
1240:
1240:
1241:
1241:
1242:
1242:
1243:
1243:
1244:
1244:
1245:
1245:
1246:
1246:
1247:
1247:
1248:
1248:
1249:
1249:
1250:
1250:
1251:
1251:
1252:
1252:
1253:
1253:
1254:
1254:
1255:
1255:
1256:
1256:
1257:
1257:
1258:
1258:
1259:
1259:
1260:
1260:
1261:
1261:
1262:
1262:
1263:
1263:
1264:
1264:
1265:
1265:
1266:
1266:
1267:
1267:
1268:
1268:
1269:
1269:
1270:
1270:
1271:
1271:
1272:
1272:
1273:
1273:
1274:
1274:
1275:
1275:
1276:
1276:
1277:
1277:
1278:
1278:
1279:
1279:
1280:
1280:
1281:
1281:
1282:
1282:
1283:
1283:
1284:
1284:
1285:
1285:
1286:
1286:
1287:
1287:
1288:
1288:
1289:
1289:
1290:
1290:
1291:
1291:
1292:
1292:
1293:
1293:
1294:
1294:
1295:
1295:
1296:
1296:
1297:
1297:
1298:
1298:
1299:
1299:
1300:
1300:
1301:
1301:
1302:
1302:
1303:
1303:
1304:
1304:
1305:
1305:
1306:
1306:
1307:
1307:
1308:
1308:
1309:
1309:
1310:
1310:
1311:
1311:
1312:
1312:
1313:
1313:
1314:
1314:
1315:
1315:
1316:
1316:
1317:
1317:
1318:
1318:
1319:
1319:
1320:
1320:
1321:
1321:
1322:
1322:
1323:
1323:
1324:
1324:
1325:
1325:
1326:
1326:
1327:
1327:
1328:
1328:
1329:
1329:
1330:
1330:
1331:
1331:
1332:
1332:
1333:
1333:
1334:
1334:
1335:
1335:
1336:
1336:
1337:
1337:
1338:
1338:
1339:
1339:
1340:
1340:
1341:
1341:
1342:
1342:
1343:
1343:
1344:
1344:
1345:
1345:
1346:
1346:
1347:
1347:
1348:
1348:
1349:
1349:
1350:
1350:
1351:
1351:
1352:
1352:
1353:
1353:
1354:
1354:
1355:
1355:
1356:
1356:
1357:
1357:
1358:
1358:
1359:
1359:
1360:
1360:
1361:
1361:
1362:
1362:
1363:
1363:
1364:
1364:
1365:
1365:
1366:
1366:
1367:
1367:
1368:
1368:
1369:
1369:
1370:
1370:
1371:
1371:
1372:
1372:
137
```

```

27:         self.__name = value
28:
29:     @property
30:     def age(self):
31:         """
32:             Getter untuk atribut age.
33:             Mengembalikan nilai dari __age.
34:         """
35:         return self.__age
36:
37:     @age.setter
38:     def age(self, value):
39:         """
40:             Setter untuk atribut age.
41:             Memeriksa apakah nilai umur tidak negatif sebelum mengubah nilai
42:             age.
43:         """
44:         if value < 0:
45:             print("Umur tidak boleh negatif!")
46:         else:
47:             self.__age = value
48: # Contoh penggunaan
49: def main():
50:     # Membuat objek Person dengan nama "Alice" dan umur 30
51:     person = Person("Alice", 30)
52:     print(f"Nama: {person.name}, Umur: {person.age}")
53:
54:     # Mengubah nama dan umur melalui setter
55:     person.name = "Bob"
56:     person.age = 35
57:     print(f"Nama baru: {person.name}, Umur baru: {person.age}")
58:
59:     # Mencoba menetapkan nilai yang tidak valid untuk umur
60:     person.age = -5 # Akan memunculkan pesan error karena validasi umur
negatif
61:
62: if __name__ == "__main__":
63:     main()

```

Penjelasan Kode

1. Atribut Privat

- `self.__name` dan `self.__age` di dalam konstruktur disembunyikan agar tidak dapat diakses atau diubah langsung dari luar kelas.

2. Getter dengan `@property`

- Metode `name` dan `age` yang didekorasi dengan `@property` berfungsi sebagai getter untuk mengembalikan nilai atribut privat.
- Contoh: `person.name` akan mengembalikan nilai `__name` tanpa mengaksesnya secara langsung.

3. Setter dengan `@.setter`

- Metode `name` dan `age` dilengkapi dengan setter yang didekorasi dengan `@name.setter` dan `@age.setter`.
- Setter digunakan untuk melakukan validasi (misalnya, nama tidak boleh kosong dan umur tidak boleh negatif) sebelum mengubah nilai atribut privat.

4. Contoh Penggunaan

- Objek dibuat dengan parameter awal.
- Akses dan modifikasi atribut dilakukan melalui property, sehingga jika ada perubahan yang tidak memenuhi validasi, pesan error akan ditampilkan.

Kode ini memberikan gambaran bagaimana penggunaan **getter**, **setter**, dan dekorator **@property** membantu dalam mengelola dan memvalidasi akses ke atribut privat dalam sebuah kelas.

E. Hasil Praktikum

Lengkapi hasil tabel praktikum berikut:

No.	Nama Praktikum	Hasil Praktikum
1	Praktikum 01: Enkapsulasi pada kelas Bank Account	
2	Praktikum 02: Enkapsulasi pada kelas Employee	
3	Praktikum 03: Kelas dan Objek Praktikum 03: Konstruktor dan Destruktor Sederhana	
4	Praktikum 04: Konstruktor dan Destruktor program FileLogger	
5	Praktikum 05: Properti "self" pada kelas	
6	Praktikum 06: Menggunakan getter, setter, dan dekorator @property untuk mengelola akses atribut	

F. Penugasan

1. Kumpulkan Laporan Praktikum dari jobsheet ini dalam bentuk Microsoft word sesuai dengan format jobsheet praktikum dan dikumpulkan di web LMS. (JANGAN DALAM BENTUK PDF)
2. Kumpulkan luaran kode praktikum dalam bentuk ipynb yang sudah diunggah pada akun github masing-masing. Lampirkan tautan github yang sudah diunggah melalui laman LMS.

3. Buat Program:

- ✓ **Buatlah sebuah kelas Python bernama Student.**
 - Kelas ini harus memiliki atribut privat: `__name`, `__score`, dan `__grade`.
 - Tambahkan konstruktor (`__init__`) untuk menginisialisasi name dan score.
- ✓ **Tambahkan getter dan setter menggunakan `@property` untuk atribut name dan score.**
 - Validasi score: harus berada di antara 0 hingga 100.
 - Jika nilai score berubah, maka grade juga harus di-update secara otomatis (gunakan fungsi setter atau method internal).
 - Rentang nilai dan grade:
 - 90–100 → A
 - 80–89 → B
 - 70–79 → C
 - 60–69 → D
 - <60 → E

✓ **Tambahkan method show_info() yang menampilkan informasi lengkap mahasiswa:**

Nama Mahasiswa: <name>

Nilai: <score>

Grade: <grade>

✓ **Tambahkan destruktur __del__ yang mencetak:**

Data mahasiswa <name> telah dihapus dari sistem.

✓ **Buat Kelas Diagram program yang dibuat**

Contoh output yang diharapkan:

```
Nama Mahasiswa: Siti
```

```
Nilai: 87
```

```
Grade: B
```

```
Nilai diubah...
```

```
Nama Mahasiswa: Siti
```

```
Nilai: 93
```

```
Grade: A
```

```
Data mahasiswa Siti telah dihapus dari sistem.
```

G. Kesimpulan

mahasiswa telah mempelajari dan mengimplementasikan konsep dasar **Object-Oriented Programming (OOP)** dalam Python secara praktis. Konsep penting seperti **kelas**, **konstruktor** (`__init__`), **destruktur** (`__del__`), **penggunaan** `self`, serta mekanisme pengelolaan atribut menggunakan **getter**, **setter**, dan **dekorator** `@property` telah diterapkan dalam konteks dunia nyata, seperti pengelolaan data mahasiswa. Dengan menggunakan `self`, mahasiswa dapat memahami bahwa setiap objek memiliki data (atribut) dan perilaku (method) yang bersifat unik dan independen. Penggunaan `@property` juga membantu melindungi data agar tidak dimodifikasi sembarangan, sekaligus memberikan cara yang lebih aman dan terkontrol dalam mengakses atribut privat. Secara keseluruhan, penugasan ini melatih mahasiswa untuk **membangun program yang terstruktur, terenkapsulasi, dan mudah dikembangkan** — sebuah fondasi penting dalam membangun sistem perangkat lunak skala kecil hingga besar.

H. Daftar Pustaka

1. Lutz, M. (2013). *Learning Python*. O'Reilly Media.
2. Guttag, J. V. (2016). *Introduction to Computation and Programming Using Python*. MIT Press.
3. Python Software Foundation. *Python 3 Documentation*. <https://docs.python.org/3/>