

IDENTITAS MATERI

Elemen	Pemrograman Berbasis Teks, Grafis dan Multimedia
Kelas / Fase	XII / F
Capaian Pembelajaran (CP)	Pada akhir fase F, murid memiliki kemampuan menerapkan perintah eksekusi bahasa pemrograman yang mengimplementasikan pemrograman terstruktur, pemrograman berorientasi objek lanjutan, pemodelan perangkat lunak, pemrograman antarmuka GUI (Graphical User Interface), dan pemanfaatan pustaka (library).
Tujuan Pembelajaran (ATP)	<ul style="list-style-type: none">2.1 Menerapkan perintah eksekusi bahasa pemrograman yang mengimplementasikan pemrograman terstruktur2.2 Menerapkan perintah eksekusi bahasa pemrograman yang mengimplementasikan pemrograman berorientasi objek lanjutan2.3 Menerapkan pemodelan perangkat lunak2.4 Menerapkan pemrograman antarmuka GUI (Graphical User Interface)2.5 Melakukan pemanfaatan pustaka (library) pada proyek pengembangan perangkat lunak

PENDAHULUAN

Perkembangan teknologi informasi telah mengubah cara manusia membangun dan menggunakan perangkat lunak (software). Di era industri 4.0 dan 5.0, seorang pengembang perangkat lunak tidak cukup hanya mengetahui cara menulis kode program. Ia perlu memahami konsep pemrograman yang terstruktur, mampu merancang sistem menggunakan pemodelan yang tepat, membuat antarmuka yang ramah pengguna, serta memanfaatkan berbagai pustaka atau library yang tersedia untuk mempercepat proses pengembangan.

Materi pembelajaran ini dirancang untuk membekali peserta didik dengan kemampuan komprehensif dalam pengembangan perangkat lunak modern. Mulai dari pemrograman terstruktur sebagai fondasi logika pemrograman, lanjut ke pemrograman berorientasi objek tingkat lanjut yang menjadi standar industri, hingga pemodelan perangkat lunak menggunakan UML, pembuatan antarmuka GUI, dan pemanfaatan pustaka untuk proyek nyata.

Melalui materi ini, peserta didik diharapkan dapat mengembangkan kemampuan berpikir komputasional, merancang solusi perangkat lunak yang efisien, dan siap menghadapi tantangan dunia kerja di bidang teknologi informasi dan pengembangan perangkat lunak.

ATP 2.1 Menerapkan perintah eksekusi bahasa pemrograman yang mengimplementasikan pemrograman terstruktur

BAB 1: PEMROGRAMAN TERSTRUKTUR

1.1 Pengertian Pemrograman Terstruktur

Pemrograman terstruktur (structured programming) adalah paradigma atau pendekatan pemrograman yang mengorganisasikan kode program ke dalam blok-blok terstruktur menggunakan tiga struktur kendali utama: urutan (sequence), pemilihan (selection), dan pengulangan (repetition/iteration). Konsep ini diperkenalkan oleh Edsger W. Dijkstra pada akhir tahun 1960-an sebagai solusi atas kompleksitas program yang tidak terorganisasi dengan baik.

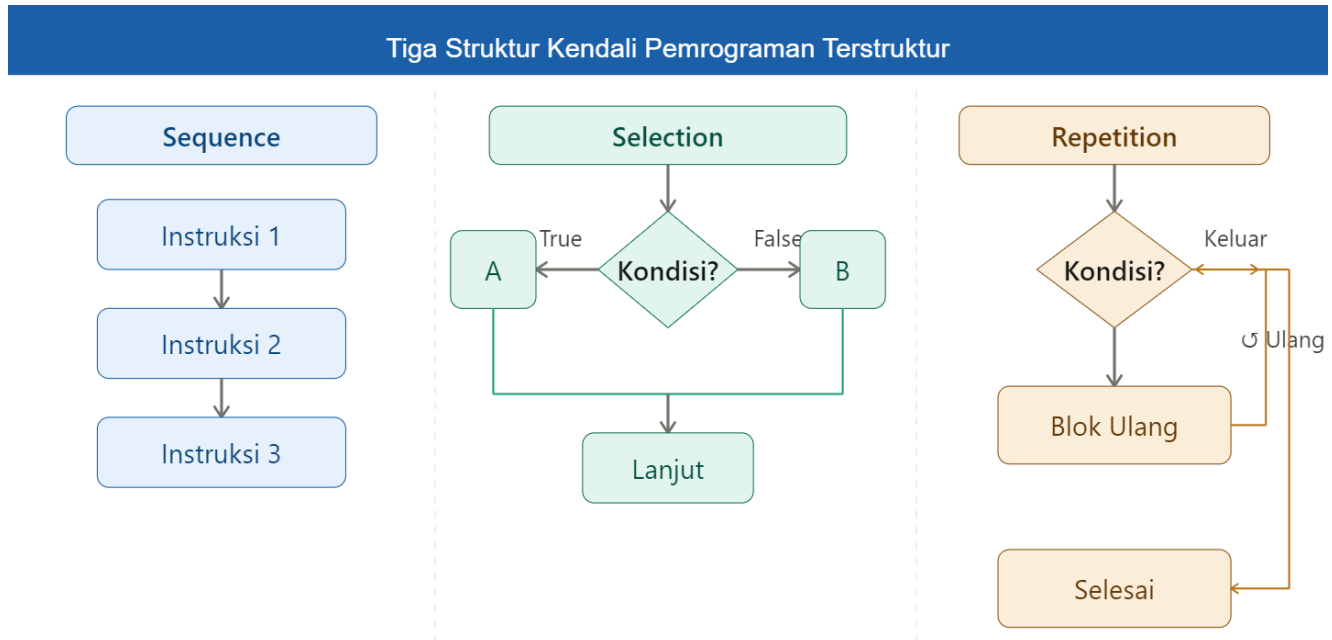
Tujuan utama pemrograman terstruktur adalah membuat kode program yang lebih mudah dipahami, dirawat, dan diuji. Dengan struktur yang jelas, kesalahan (bug) lebih mudah ditemukan dan diperbaiki karena alur logika program mengikuti pola yang dapat diprediksi.

Pengertian Pemrograman Terstruktur

Pemrograman Terstruktur adalah metode penulisan program komputer yang menggunakan tiga struktur kendali utama:

- Sequence (Urutan): instruksi dieksekusi satu per satu secara berurutan dari atas ke bawah.
- Selection (Pemilihan): program membuat keputusan berdasarkan kondisi tertentu (if-else, switch).
- Repetition (Pengulangan): instruksi diulang selama kondisi tertentu terpenuhi (for, while, do-while).

1.2 Konsep Dasar Pemrograman Terstruktur



Gambar 1.1 Ilustrasi Tiga Struktur Kendali dalam Pemrograman Terstruktur

1.2.1 Struktur Sequence (Urutan)

Sequence adalah struktur paling dasar dalam pemrograman. Instruksi-instruksi dieksekusi secara berurutan dari atas ke bawah, satu per satu. Setiap instruksi selesai dieksekusi sebelum instruksi berikutnya dimulai. Contoh paling sederhana adalah program menghitung luas persegi panjang yang pertama meminta input panjang, lalu lebar, kemudian menghitung luas, dan terakhir menampilkan hasil.

```
# Contoh Sequence dalam Python
panjang = float(input('Masukkan panjang: '))
lebar   = float(input('Masukkan lebar: '))
luas    = panjang * lebar
print(f'Luas persegi panjang: {luas}')
```

Contoh 1.1 - Struktur Sequence dalam Python

1.2.2 Struktur Selection (Pemilihan)

Selection atau seleksi memungkinkan program untuk membuat keputusan. Program akan mengevaluasi suatu kondisi (ekspresi boolean) dan memilih jalur eksekusi yang berbeda berdasarkan hasil evaluasi tersebut. Dalam Python, struktur seleksi dinyatakan dengan kata kunci `if`, `elif`, dan `else`.

```
# Contoh Selection dalam Python
nilai = int(input('Masukkan nilai ujian (0-100): '))

if nilai >= 90:
    print('Grade: A - Sangat Baik')
```

```
elif nilai >= 80:
    print('Grade: B - Baik')
elif nilai >= 70:
    print('Grade: C - Cukup')
elif nilai >= 60:
    print('Grade: D - Kurang')
else:
    print('Grade: E - Tidak Lulus')
```

Contoh 1.2 - Struktur Selection (if-elif-else) dalam Python

1.2.3 Struktur Repetition (Pengulangan)

Repetition atau pengulangan memungkinkan sekumpulan instruksi dieksekusi berkali-kali. Terdapat tiga jenis pengulangan utama: for loop (pengulangan dengan jumlah iterasi yang diketahui), while loop (pengulangan selama kondisi terpenuhi), dan do-while (minimal satu kali eksekusi). Python menggunakan for dan while sebagai struktur pengulangan.

```
# Contoh Repetition - For Loop dalam Python
# Menampilkan tabel perkalian 5
for i in range(1, 11):
    print(f'5 x {i} = {5 * i}')

# Contoh While Loop
total = 0
angka = 1
while angka <= 100:
    total += angka
    angka += 1
print(f'Total 1 sampai 100 = {total}')
```

Contoh 1.3 - Struktur Repetition (for & while) dalam Python

1.3 Prosedur dan Fungsi

Salah satu prinsip penting dalam pemrograman terstruktur adalah dekomposisi, yaitu memecah program besar menjadi unit-unit yang lebih kecil dan dapat digunakan ulang (reusable). Unit-unit ini disebut prosedur (procedure) atau fungsi (function). Dalam Python, keduanya dinyatakan dengan kata kunci def.

```
# Contoh Fungsi dalam Python
def hitung_faktorial(n):
    '''Menghitung nilai faktorial dari n'''
    if n == 0 or n == 1:
        return 1
    hasil = 1
    for i in range(2, n + 1):
        hasil *= i
    return hasil
```

```
# Memanggil fungsi
print(hitung_faktorial(5))    # Output: 120
print(hitung_faktorial(10))  # Output: 3628800
```

Contoh 1.4 - Fungsi rekursif dan iteratif dalam Python

1.4 Prinsip-Prinsip Pemrograman Terstruktur

Prinsip	Penjelasan	Contoh Penerapan
Single Entry Single Exit	Setiap blok kode memiliki satu titik masuk dan satu titik keluar	Menghindari goto statement
Top-Down Design	Program dirancang dari gambaran umum ke detail	Membuat pseudocode sebelum koding
Modularisasi	Memecah program menjadi modul-modul kecil	Membuat fungsi terpisah untuk setiap tugas
Abstraksi	Menyembunyikan detail implementasi	Fungsi dengan nama yang deskriptif
Stepwise Refinement	Mengembangkan solusi secara bertahap	Iterasi dalam pengembangan modul

Tabel 1.1 - Prinsip-Prinsip Pemrograman Terstruktur

1.5 Implementasi Pemrograman Terstruktur dalam Proyek

Berikut adalah contoh implementasi pemrograman terstruktur yang lebih kompleks: program manajemen nilai siswa yang menggunakan fungsi-fungsi terstruktur untuk mengelola data.

```
# Sistem Manajemen Nilai Siswa - Pemrograman Terstruktur

def tambah_nilai(data_siswa, nama, nilai):
    data_siswa.append({'nama': nama, 'nilai': nilai})

def hitung_rata_rata(data_siswa):
    if not data_siswa:
        return 0
    total = sum(s['nilai'] for s in data_siswa)
    return total / len(data_siswa)

def tampilkan_laporan(data_siswa):
    print('=== LAPORAN NILAI SISWA ===')
    for siswa in data_siswa:
        status = 'Lulus' if siswa['nilai'] >= 75 else 'Tidak Lulus'
        print(f"{siswa['nama']:20} | {siswa['nilai']:3} | {status}")
    print(f'Rata-rata kelas: {hitung_rata_rata(data_siswa):.2f}')
```

```
# Program utama (main program)
data = []
tambah_nilai(data, 'Budi Santoso', 88)
tambah_nilai(data, 'Ani Rahayu', 92)
tambah_nilai(data, 'Candra Putra', 70)
tampilkan_laporan(data)
```

Contoh 1.5 - Proyek Manajemen Nilai menggunakan Pemrograman Terstruktur

Sumber Referensi — Bab 1

- Dijkstra, E. W. (1968). Go To Statement Considered Harmful. Communications of the ACM, 11(3), 147-148.
- Sebesta, R. W. (2019). Concepts of Programming Languages (12th ed.). Pearson Education.
- Deitel, P., & Deitel, H. (2021). Python How to Program (3rd ed.). Pearson Education.
- Hetland, M. L. (2017). Python Algorithms: Mastering Basic Algorithms in the Python Language (2nd ed.). Apress.

ATP 2.2 Menerapkan perintah eksekusi bahasa pemrograman yang mengimplementasikan pemrograman berorientasi objek lanjutan

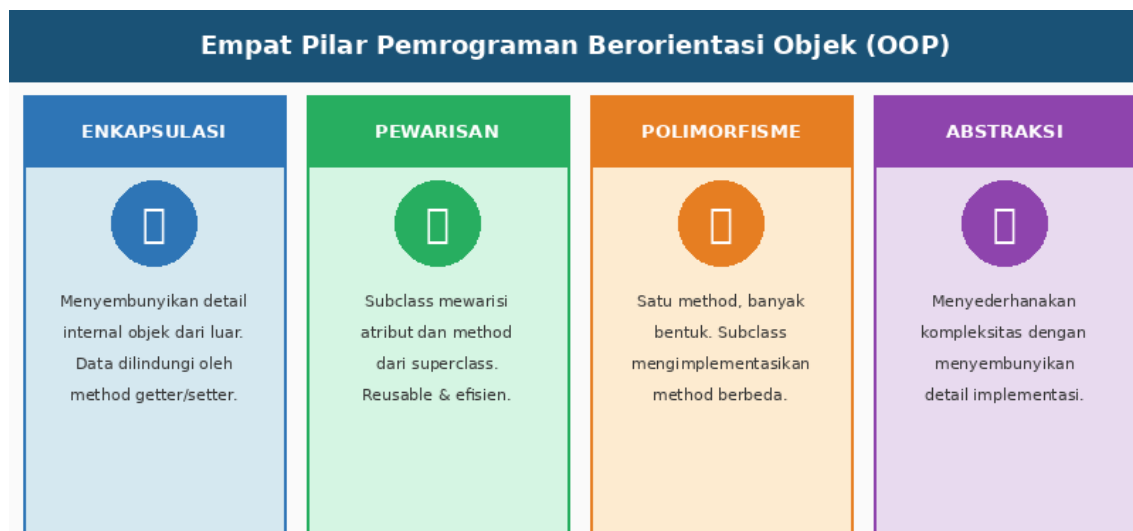
BAB 2: PEMROGRAMAN BERORIENTASI OBJEK LANJUTAN

2.1 Tinjauan Ulang Konsep Dasar OOP

Pemrograman Berorientasi Objek (Object-Oriented Programming / OOP) adalah paradigma pemrograman yang mengorganisasikan program sebagai kumpulan objek yang saling berinteraksi. Setiap objek merupakan instansi dari sebuah kelas (class) yang mendefinisikan atribut (data/properti) dan metode (perilaku/fungsi) yang dimilikinya.

Konsep Dasar OOP	Penjelasan Singkat
Class	Blueprint atau cetakan untuk membuat objek
Object	Instansi nyata dari sebuah class
Attribute	Data atau properti yang dimiliki objek
Method	Fungsi atau perilaku yang dimiliki objek
Constructor	Metode khusus untuk inisialisasi objek (<code>__init__</code> di Python)

Tabel 2.1 - Konsep Dasar OOP



Gambar 2.1 Empat Pilar Utama OOP dan Penjelasannya

2.2 Pewarisan (Inheritance) Lanjutan

Inheritance (pewarisan) adalah mekanisme di mana sebuah class (subclass/child class) mewarisi atribut dan metode dari class lain (superclass/parent class). Dalam OOP lanjutan, kita mempelajari single inheritance (satu parent), multiple inheritance (lebih dari satu parent), dan multilevel inheritance (rantai pewarisan).


```
# Contoh Inheritance Lanjutan dalam Python

class Hewan:
    def __init__(self, nama, berat):
        self.nama = nama
        self.berat = berat

    def bergerak(self):
        return f'{self.nama} sedang bergerak'

    def __str__(self):
        return f'Hewan: {self.nama} ({self.berat} kg)'

class Mamalia(Hewan):          # Single Inheritance
    def __init__(self, nama, berat, jenis_susu):
        super().__init__(nama, berat)    # Memanggil constructor parent
        self.jenis_susu = jenis_susu

    def menyusui(self):
        return f'{self.nama} menyusui anaknya'

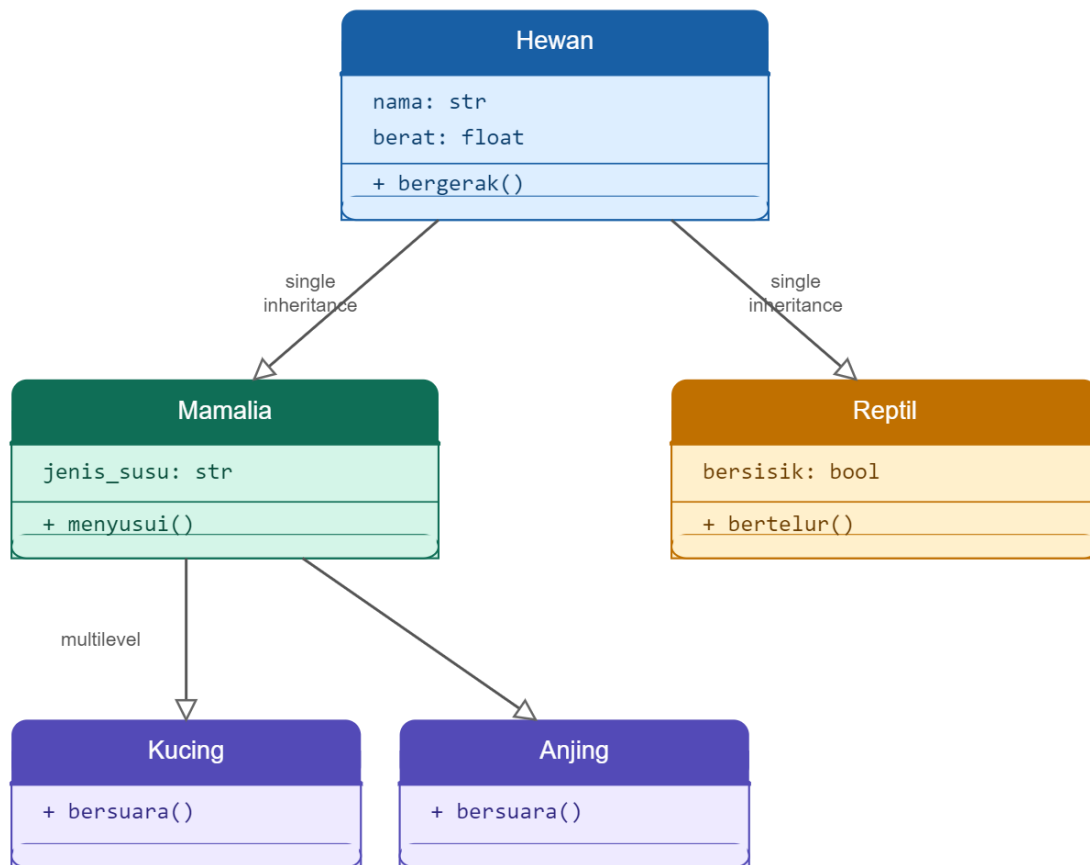
class Kucing(Mamalia):        # Multilevel Inheritance
    def __init__(self, nama, berat):
        super().__init__(nama, berat, 'susu kucing')

    def bersuara(self):
        return f'{self.nama} berkata: Meow!'

# Penggunaan
kucing = Kucing('Mimi', 4.5)
print(kucing)                 # Hewan: Mimi (4.5 kg)
print(kucing.bergerak())      # Mimi sedang bergerak
print(kucing.menyusui())      # Mimi menyusui anaknya
print(kucing.bersuara())      # Mimi berkata: Meow!
```

Contoh 2.1 - Implementasi Multilevel Inheritance dalam Python

Diagram Pewarisan (Inheritance) — Multilevel & Multiple



Gambar 2.2 Diagram Hubungan Pewarisan Kelas (Inheritance Hierarchy)

2.3 Enkapsulasi (Encapsulation)

Enkapsulasi adalah prinsip OOP yang menyembunyikan detail implementasi internal sebuah objek dari dunia luar. Ini dilakukan dengan mengontrol akses ke atribut dan metode menggunakan access modifier. Tujuannya adalah melindungi integritas data dan menyederhanakan antarmuka yang diekspos ke pengguna kelas.

```

# Contoh Enkapsulasi dalam Python

class RekeningBank:
    def __init__(self, nomor_rekening, pemilik, saldo_awal):
        self.__nomor_rekening = nomor_rekening # Private (__)
        self.__pemilik = pemilik
        self.__saldo = saldo_awal
        self._history = [] # Protected (_)

    # Getter - mengambil nilai atribut private
    @property
    def saldo(self):

```

```

        return self.__saldo

# Method untuk validasi sebelum mengubah data
def setor(self, jumlah):
    if jumlah <= 0:
        raise ValueError('Jumlah setoran harus positif')
    self.__saldo += jumlah
    self._history.append(f'Setor: +Rp{jumlah:,.0f}')
    return True

def tarik(self, jumlah):
    if jumlah > self.__saldo:
        raise ValueError('Saldo tidak mencukupi')
    self.__saldo -= jumlah
    self._history.append(f'Tarik: -Rp{jumlah:,.0f}')
    return True

rek = RekeningBank('1234567890', 'Budi', 1000000)
rek.setor(500000)
rek.tarik(200000)
print(f'Saldo: Rp{rek.saldo:,.0f}') # Rp1,300,000

```

Contoh 2.2 - Enkapsulasi dengan Access Modifier dalam Python

2.4 Polimorfisme (Polymorphism)

Polimorfisme berarti 'banyak bentuk'. Dalam OOP, polimorfisme memungkinkan objek dari kelas yang berbeda untuk diakses melalui antarmuka yang sama. Terdapat dua jenis polimorfisme utama: method overriding (mendefinisikan ulang method parent di subclass) dan method overloading (method dengan nama sama tetapi parameter berbeda).

```

# Contoh Polimorfisme - Method Overriding

class Bentuk:
    def luas(self):
        raise NotImplementedError('Subclass harus mengimplementasikan luas()')

    def keliling(self):
        raise NotImplementedError('Subclass harus mengimplementasikan keliling()')

    def info(self):
        return f'Luas: {self.luas():.2f} | Keliling: {self.keliling():.2f}'

class Lingkaran(Bentuk):
    def __init__(self, jari_jari):
        self.r = jari_jari

    def luas(self): # Override
        return 3.14159 * self.r ** 2

```

```

    def keliling(self): # Override
        return 2 * 3.14159 * self.r

class Persegi(Bentuk):
    def __init__(self, sisi):
        self.s = sisi

    def luas(self): # Override
        return self.s ** 2

    def keliling(self): # Override
        return 4 * self.s

# Polimorfisme dalam aksi
bentuk_list = [Lingkaran(7), Persegi(5), Lingkaran(3)]
for b in bentuk_list:
    print(b.info()) # Metode yang sama, hasil berbeda

```

Contoh 2.3 - Polimorfisme dengan Method Overriding

2.5 Abstraksi dan Abstract Class

Abstraksi adalah prinsip OOP yang memungkinkan kita mendefinisikan antarmuka (interface) tanpa perlu menentukan detail implementasinya. Dalam Python, abstraksi diimplementasikan menggunakan modul abc (Abstract Base Classes). Sebuah abstract class tidak dapat diinstansiasi langsung; ia hanya berfungsi sebagai blueprint bagi subclass.

```

from abc import ABC, abstractmethod

class Kendaraan(ABC): # Abstract Base Class
    def __init__(self, merek, tahun):
        self.merek = merek
        self.tahun = tahun

    @abstractmethod # Method abstrak - WAJIB diimplementasikan subclass
    def bahan_bakar(self):
        pass

    @abstractmethod
    def kecepatan_maks(self):
        pass

    def info(self): # Concrete method - bisa diwarisi langsung
        return (f'{self.merek} ({self.tahun}) | '
                f'BBM: {self.bahan_bakar()} | '
                f'Maks: {self.kecepatan_maks()} km/h')

```

```

class Mobil(Kendaraan):
    def bahan_bakar(self): return 'Bensin / Diesel'
    def kecepatan_maks(self): return 200

class SepedaMotor(Kendaraan):
    def bahan_bakar(self): return 'Bensin'
    def kecepatan_maks(self): return 140

mobil = Mobil('Toyota Avanza', 2023)
motor = SepedaMotor('Honda Vario', 2024)
print(mobil.info())
print(motor.info())

```

Contoh 2.4 - Abstract Base Class dalam Python

2.6 Design Patterns Dasar dalam OOP

Design pattern adalah solusi yang sudah terbukti untuk masalah desain perangkat lunak yang sering muncul. Berikut adalah dua pola desain yang paling sering digunakan oleh pemula:

Design Pattern	Kegunaan dan Karakteristik
Singleton Pattern	Memastikan sebuah class hanya memiliki satu instansi di seluruh program. Sering digunakan untuk koneksi database, logger, dan konfigurasi aplikasi.
Factory Pattern	Menyediakan antarmuka untuk membuat objek tanpa menentukan class spesifiknya. Berguna saat jenis objek ditentukan saat runtime.
Observer Pattern	Mendefinisikan ketergantungan satu-ke-banyak antara objek, sehingga saat satu objek berubah state, semua dependennya diberitahu otomatis.
Strategy Pattern	Mendefinisikan sekumpulan algoritma, mengenkapsulasi masing-masing, dan membuatnya dapat dipertukarkan.

Tabel 2.2 - Design Patterns Dasar dalam OOP

Sumber Referensi — Bab 2

- Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley.
- Lutz, M. (2019). Learning Python (5th ed.). O'Reilly Media.
- Phillips, D. (2018). Python 3 Object-Oriented Programming (3rd ed.). Packt Publishing.

- Martin, R. C. (2008). Clean Code: A Handbook of Agile Software Craftsmanship. Prentice Hall.

BAB 3: PEMODELAN PERANGKAT LUNAK

3.1 Pengertian Pemodelan Perangkat Lunak

Pemodelan perangkat lunak (software modeling) adalah proses membuat representasi abstrak dari sistem perangkat lunak yang akan dibangun. Model ini berfungsi sebagai cetak biru (blueprint) yang membantu pengembang memahami struktur sistem, hubungan antar komponen, dan alur kerja sistem sebelum mulai menulis kode program.

Sama seperti seorang arsitek yang membuat gambar denah rumah sebelum membangun, seorang pengembang perangkat lunak membuat diagram dan model sebelum mengimplementasikan sistem. Pemodelan membantu mengidentifikasi masalah desain lebih awal, sebelum menjadi mahal untuk diperbaiki.

Manfaat Pemodelan Perangkat Lunak

- Komunikasi: Model memudahkan komunikasi antara pengembang, analis, dan pelanggan.
- Dokumentasi: Model berfungsi sebagai dokumentasi sistem yang dapat dirujuk di masa mendatang.
- Deteksi Masalah Dini: Masalah desain lebih mudah ditemukan sebelum implementasi.
- Perencanaan: Model membantu estimasi waktu dan sumber daya yang dibutuhkan.
- Pemeliharaan: Sistem yang terdokumentasi lebih mudah dipelihara dan dikembangkan.

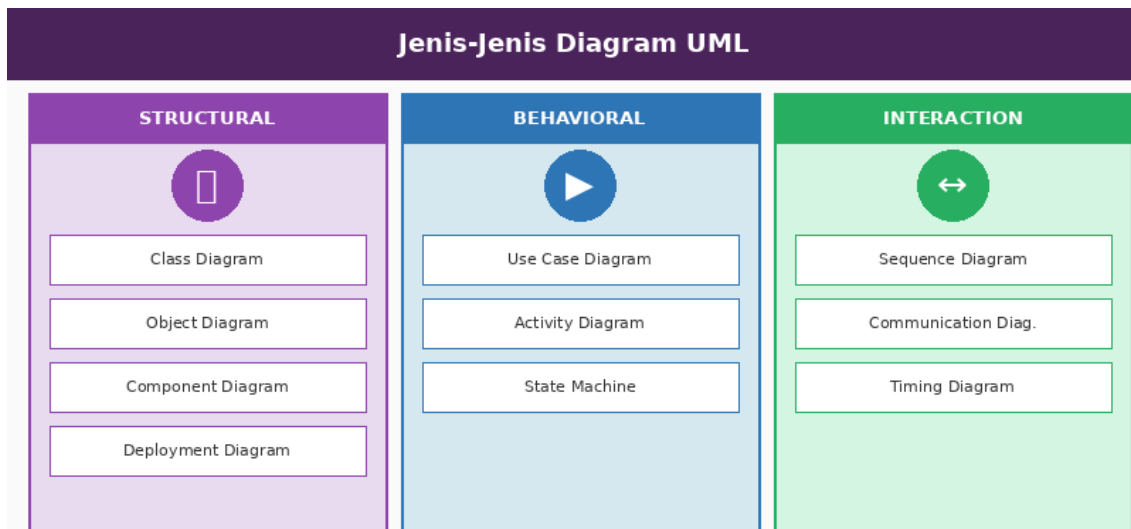
3.2 UML (Unified Modeling Language)

UML (Unified Modeling Language) adalah bahasa pemodelan standar yang digunakan secara luas dalam industri pengembangan perangkat lunak. UML dikembangkan oleh Grady Booch, James Rumbaugh, dan Ivar Jacobson pada tahun 1990-an dan kini distandarisasi oleh OMG (Object Management Group). UML menyediakan berbagai jenis diagram untuk memodelkan aspek yang berbeda dari sebuah sistem.

Kategori Diagram UML	Jenis-Jenis Diagram
Structural Diagrams (Struktur Statis Sistem)	Class Diagram, Object Diagram, Component Diagram, Deployment Diagram, Package Diagram

Kategori Diagram UML	Jenis-Jenis Diagram
Behavioral Diagrams (Perilaku Dinamis Sistem)	Use Case Diagram, Activity Diagram, State Machine Diagram
Interaction Diagrams (Interaksi antar Objek)	Sequence Diagram, Communication Diagram, Timing Diagram

Tabel 3.1 - Kategori dan Jenis Diagram UML



Gambar 3.1 Kategori dan Jenis Diagram dalam UML (Unified Modeling Language)

3.3 Use Case Diagram

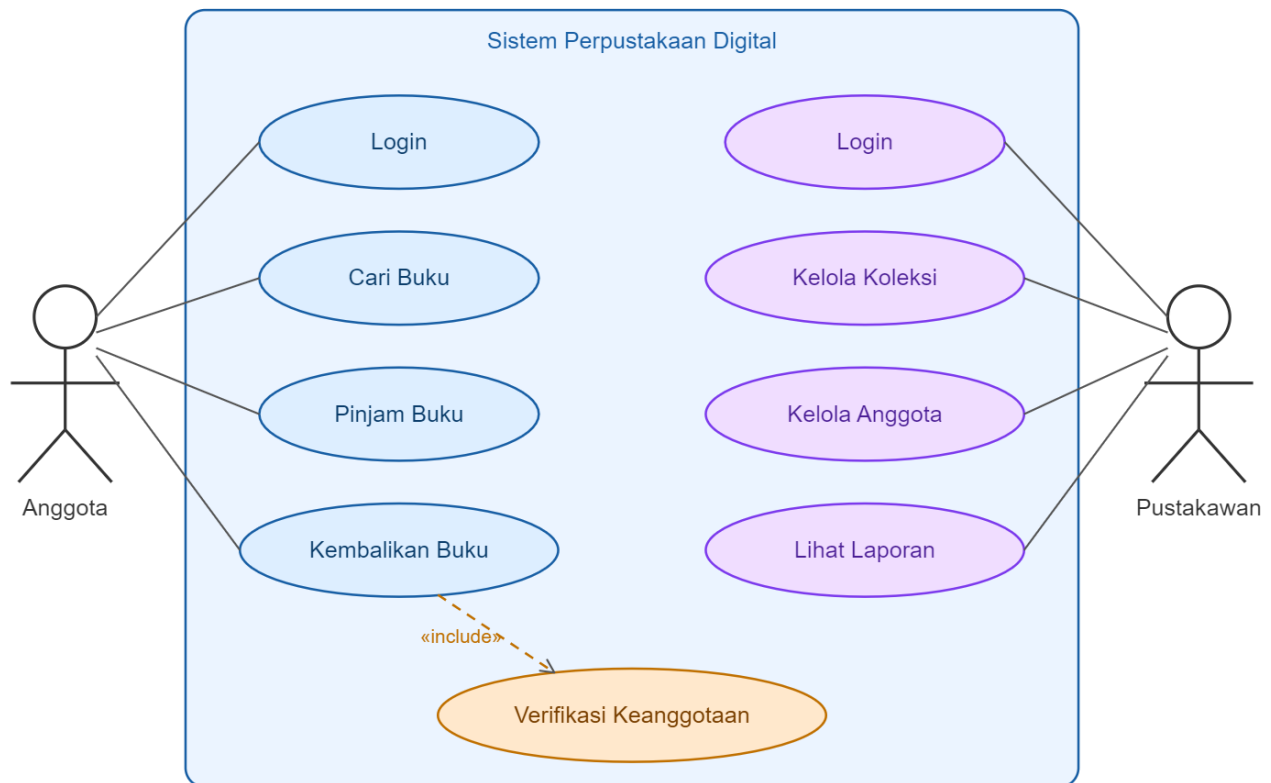
Use Case Diagram menggambarkan interaksi antara pengguna (aktor) dengan sistem. Diagram ini menunjukkan apa yang dapat dilakukan sistem dari sudut pandang pengguna, bukan bagaimana sistem melakukannya. Use Case Diagram sangat berguna untuk mengumpulkan dan mendokumentasikan kebutuhan fungsional sistem.

Elemen	Simbol	Keterangan
Actor	Gambar orang (stickman)	Pengguna atau sistem lain yang berinteraksi dengan sistem
Use Case	Oval / Ellipse	Fungsi atau layanan yang disediakan sistem
System Boundary	Kotak persegi panjang	Batas sistem yang sedang dimodelkan
Association	Garis lurus	Hubungan antara aktor dan use case
Include	Garis putus-putus <<include>>	Use case selalu memanggil use case lain

Elemen	Simbol	Keterangan
Extend	Garis putus-putus <<extend>>	Use case opsional diperluas dari use case lain

Tabel 3.2 - Elemen-Elemen Use Case Diagram

Contoh Use Case Diagram — Sistem Perpustakaan Digital



Gambar 3.2 Use Case Diagram untuk Sistem Perpustakaan Digital

Contoh Use Case Diagram untuk Sistem Perpustakaan Digital: sistem ini memiliki aktor Anggota dan Pustakawan. Anggota dapat melakukan use case: Login, Cari Buku, Pinjam Buku (include: Verifikasi Keanggotaan), dan Kembalikan Buku. Pustakawan dapat melakukan: Login, Kelola Koleksi Buku, Kelola Data Anggota, dan Lihat Laporan Peminjaman.

3.4 Class Diagram

Class Diagram menggambarkan struktur statis sistem dengan menunjukkan kelas-kelas yang ada, atributnya, metodenya, dan hubungan antar kelas. Class Diagram adalah diagram yang paling sering digunakan dalam pengembangan sistem berorientasi objek karena langsung berkorespondensi dengan kode program.

Hubungan Antar Kelas	Simbol	Makna
Association	Garis lurus	Kelas A memiliki referensi ke kelas B
Aggregation	Garis dengan diamond terbuka	Hubungan 'has-a' yang lemah (bagian bisa berdiri sendiri)
Composition	Garis dengan diamond terisi	Hubungan 'has-a' yang kuat (bagian tidak bisa berdiri sendiri)
Inheritance	Garis dengan panah segitiga terbuka	Hubungan 'is-a' (pewarisan)
Dependency	Garis putus-putus dengan panah terbuka	Kelas A bergantung pada kelas B
Realization	Garis putus-putus dengan segitiga terbuka	Kelas mengimplementasikan interface

Tabel 3.3 - Hubungan Antar Kelas dalam Class Diagram

3.5 Sequence Diagram

Sequence Diagram menggambarkan interaksi antar objek dalam urutan waktu (kronologis). Diagram ini sangat berguna untuk memodelkan alur kerja suatu fitur atau use case secara detail. Sequence Diagram menunjukkan pesan apa yang dikirim antar objek dan dalam urutan apa.

Komponen utama Sequence Diagram meliputi: Actor (pihak yang memulai interaksi), Lifeline (garis putus-putus vertikal yang mewakili objek), Activation Bar (kotak di atas lifeline menunjukkan objek sedang aktif), Message (panah horizontal yang mewakili pemanggilan metode atau pengiriman data), dan Return Message (panah putus-putus kembali ke pemanggil).

3.6 Activity Diagram

Activity Diagram menggambarkan alur kerja atau proses bisnis dalam sebuah sistem. Diagram ini mirip dengan flowchart tetapi dengan notasi UML yang lebih standar. Activity Diagram sangat berguna untuk memodelkan proses bisnis yang kompleks, algoritma, dan alur kerja paralel.

Simbol	Nama	Fungsi
Lingkaran hitam penuh	Initial Node	Titik awal aktivitas

Simbol	Nama	Fungsi
Lingkaran ganda (lingkaran hitam dalam)	Final Node	Titik akhir aktivitas
Persegi panjang dengan sudut membulat	Action	Langkah atau aktivitas tunggal
Belah ketupat (diamond)	Decision Node	Titik pengambilan keputusan (percabangan)
Garis tebal horizontal	Fork/Join Bar	Pemecahan atau penggabungan alur paralel
Kotak/swim lane	Partition	Mengelompokkan aktivitas berdasarkan aktor/komponen

Tabel 3.4 - Simbol-Simbol dalam Activity Diagram

3.7 Tools Pemodelan Perangkat Lunak

Tools	Keterangan dan Keunggulan
draw.io (diagrams.net)	Tools gratis berbasis web untuk membuat semua jenis diagram UML. Tidak perlu instalasi, dapat disimpan ke Google Drive.
StarUML	Aplikasi desktop khusus UML yang populer di kalangan pelajar dan profesional. Mendukung pembuatan semua jenis diagram UML.
PlantUML	Tools berbasis teks untuk membuat diagram UML dari kode. Dapat diintegrasikan dengan IDE seperti VS Code.
Lucidchart	Platform berbasis web dengan fitur kolaborasi real-time. Ideal untuk kerja tim.
Visual Paradigm	Tools profesional dengan fitur lengkap termasuk code generation dari diagram.

Tabel 3.5 - Tools Pemodelan Perangkat Lunak

Sumber Referensi — Bab 3

- Fowler, M. (2003). UML Distilled: A Brief Guide to the Standard Object Modeling Language (3rd ed.). Addison-Wesley.

• Pressman, R. S., & Maxim, B. R. (2020). Software Engineering: A Practitioner's Approach (9th ed.). McGraw-Hill.

• Rumbaugh, J., Jacobson, I., & Booch, G. (2005). The Unified Modeling Language Reference Manual (2nd ed.). Addison-Wesley.

• Sommerville, I. (2016). Software Engineering (10th ed.). Pearson Education.

ATP 2.4 Menerapkan pemrograman antarmuka GUI (Graphical User Interface)

BAB 4: PEMROGRAMAN ANTARMUKA GUI

4.1 Pengertian GUI

GUI (Graphical User Interface) adalah jenis antarmuka pengguna yang memungkinkan interaksi dengan perangkat lunak melalui elemen grafis seperti jendela (window), ikon, tombol, menu, dan lainnya, alih-alih perintah teks. GUI mengubah cara manusia berinteraksi dengan komputer secara fundamental, membuat komputer menjadi lebih mudah digunakan oleh semua kalangan.

Dalam konteks pemrograman, GUI programming adalah proses membuat antarmuka pengguna grafis untuk aplikasi desktop. Sebuah aplikasi GUI terdiri dari widget atau komponen antarmuka yang ditata dalam layout tertentu, dan setiap komponen dapat merespons event (kejadian) seperti klik mouse, pengetikan keyboard, atau penutupan jendela.

Konsep Penting dalam GUI Programming

- Widget: Komponen antarmuka seperti tombol, label, kotak teks, checkbox.
- Container: Widget yang dapat memuat widget lain (Frame, Window, Panel).
- Layout Manager: Sistem yang mengatur posisi dan ukuran widget di dalam container.
- Event: Aksi pengguna seperti klik, ketik, scroll, atau pergerakan mouse.
- Event Handler / Callback: Fungsi yang dijalankan sebagai respons terhadap event.
- Event Loop: Mekanisme yang terus mendengarkan dan memproses event dari pengguna.

4.2 Framework GUI untuk Python

Framework	Keunggulan	Kekurangan
Tkinter	Bawaan Python, mudah dipelajari, dokumentasi lengkap	Tampilan kurang modern secara default
PyQt5/PyQt6	Tampilan profesional, fitur lengkap, lintas platform	Kurva belajar lebih curam, lisensi komersial
wxPython	Tampilan native OS, performa bagus	Setup lebih kompleks

Framework	Keunggulan	Kekurangan
Kivy	Mendukung layar sentuh dan mobile, modern	Tidak cocok untuk aplikasi desktop tradisional
CustomTkinter	Tampilan modern berbasis Tkinter, mudah digunakan	Library pihak ketiga, perlu instalasi terpisah

Tabel 4.1 - Perbandingan Framework GUI untuk Python

4.3 Memulai dengan Tkinter

Tkinter adalah framework GUI bawaan Python yang paling banyak digunakan untuk pembelajaran. Tkinter merupakan binding Python untuk toolkit GUI Tk, yang tersedia di hampir semua instalasi Python tanpa perlu instalasi tambahan. Meski sederhana, Tkinter cukup untuk membuat aplikasi desktop yang fungsional.

```
import tkinter as tk
from tkinter import ttk, messagebox

# Membuat jendela utama aplikasi
root = tk.Tk()
root.title('Aplikasi Pertamaku')
root.geometry('400x300') # Lebar x Tinggi (pixel)
root.resizable(True, True) # Bisa diresize

# Menambahkan widget Label
label = tk.Label(root, text='Halo, Dunia!',
                  font=('Arial', 16, 'bold'),
                  fg='navy')
label.pack(pady=20)

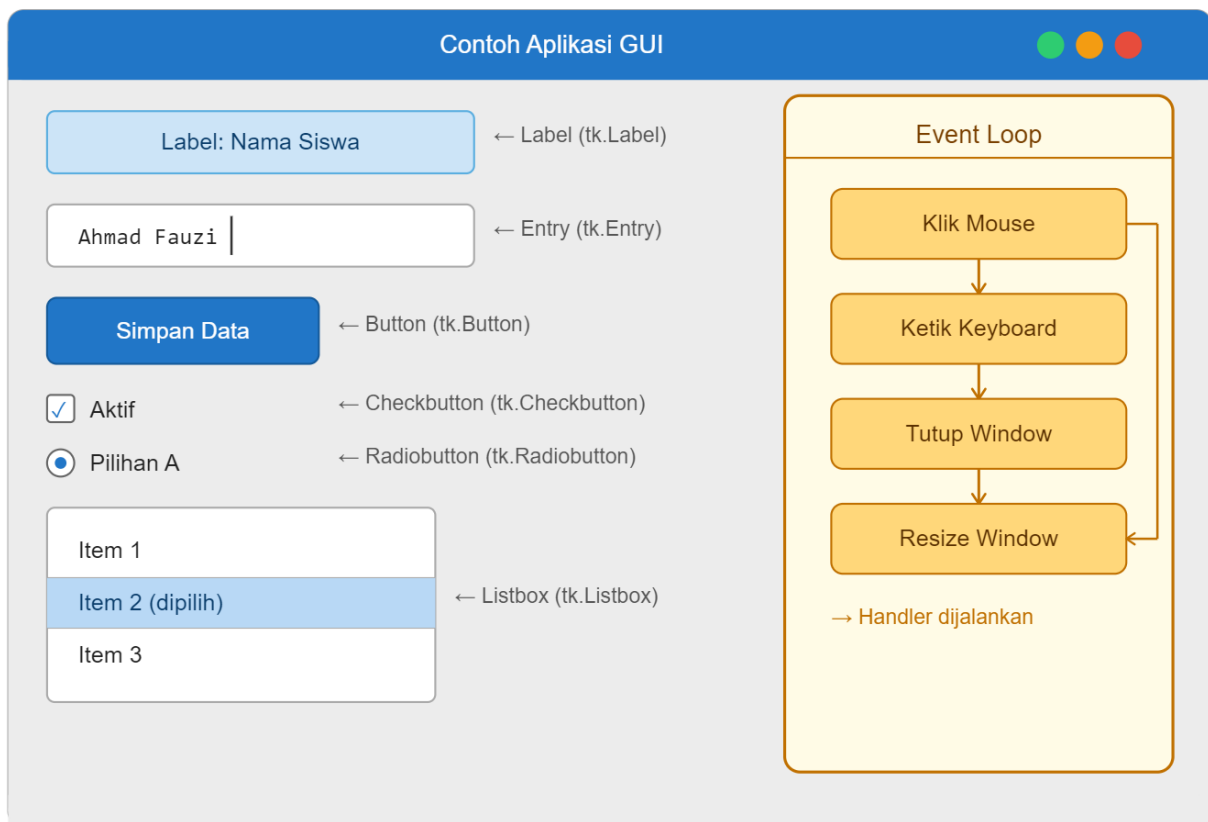
# Menambahkan widget Button
def klik_tombol():
    messagebox.showinfo('Pesan', 'Tombol diklik!')

tombol = tk.Button(root, text='Klik Saya!',
                   command=klik_tombol,
                   bg='#2E75B6', fg='white',
                   font=('Arial', 12),
                   padx=20, pady=8)
tombol.pack(pady=10)

# Memulai event loop
root.mainloop()
```

Contoh 4.1 - Aplikasi GUI Pertama dengan Tkinter

Komponen-Komponen Utama Aplikasi GUI (Tkinter)



Gambar 4.1 Widget Utama Tkinter dan Konsep Event Loop dalam GUI Programming

4.4 Widget-Widget Utama Tkinter

Widget	Kelas Tkinter	Fungsi
Label	tk.Label	Menampilkan teks atau gambar (tidak interaktif)
Button	tk.Button	Tombol yang dapat diklik untuk memicu aksi
Entry	tk.Entry	Kotak input teks satu baris
Text	tk.Text	Area teks multi-baris untuk input atau tampilan
Checkbutton	tk.Checkbutton	Kotak centang (checkbox) untuk pilihan ya/tidak
Radiobutton	tk.Radiobutton	Tombol pilihan untuk memilih satu dari beberapa opsi
Listbox	tk.Listbox	Daftar item yang dapat dipilih

Widget	Kelas Tkinter	Fungsi
Combobox	ttk.Combobox	Dropdown list dengan opsi yang dapat dipilih
Scale/Slider	tk.Scale	Slider untuk memilih nilai numerik
Frame	tk.Frame	Container untuk mengelompokkan widget lain
Canvas	tk.Canvas	Area gambar untuk grafis, garis, dan bentuk
Scrollbar	tk.Scrollbar	Scroll bar untuk widget yang dapat digulir

Tabel 4.2 - Widget-Widget Utama Tkinter

4.5 Layout Manager Tkinter

Layout manager mengontrol bagaimana widget ditempatkan di dalam container. Tkinter memiliki tiga layout manager utama: pack, grid, dan place. Pemilihan layout manager yang tepat sangat mempengaruhi kerapian dan keindahan tampilan aplikasi.

Layout Manager	Karakteristik dan Kapan Digunakan
pack()	Menempatkan widget dalam urutan horizontal atau vertikal. Cocok untuk tata letak sederhana. Parameter: side (TOP/BOTTOM/LEFT/RIGHT), fill, expand, padx, pady.
grid()	Menempatkan widget dalam format baris dan kolom seperti tabel. Cocok untuk form dengan label dan input yang sejajar. Parameter: row, column, rowspan, colspan, sticky, padx, pady.
place()	Menempatkan widget di posisi absolut (koordinat x, y). Cocok saat membutuhkan posisi pixel yang tepat. Tidak responsif terhadap resize.

Tabel 4.3 - Perbandingan Layout Manager Tkinter

4.6 Event Handling

Event handling adalah mekanisme untuk merespons aksi pengguna. Di Tkinter, kita menghubungkan event dengan fungsi handler menggunakan metode bind() atau parameter command. Event diberi nama dalam format string seperti '<Button-1>' (klik kiri), '<KeyPress>', atau '<Enter>'.

```
import tkinter as tk
```



```

def on_klik(event):
    label.config(text=f'Klik di ({event.x}, {event.y})')

def on_keyboard(event):
    label.config(text=f'Tombol: {event.char} | Kode: {event.keycode}')

def on_hover(event):
    tombol.config(bg='#1a5276')

def on_leave(event):
    tombol.config(bg='#2E75B6')

root = tk.Tk()
root.geometry('400x200')

label = tk.Label(root, text='Coba klik atau tekan tombol keyboard',
                 font=('Arial', 11))
label.pack(pady=20)

tombol = tk.Button(root, text='Area Klik', bg='#2E75B6', fg='white')
tombol.pack(pady=10)

# Binding event ke widget
tombol.bind('<Button-1>', on_klik)      # Klik kiri mouse
root.bind('<KeyPress>', on_keyboard)  # Tekanan tombol keyboard
tombol.bind('<Enter>', on_hover)      # Mouse masuk ke widget
tombol.bind('<Leave>', on_leave)      # Mouse keluar dari widget

root.mainloop()

```

Contoh 4.2 - Event Handling dalam Tkinter

4.7 Proyek GUI: Aplikasi Kalkulator

Berikut adalah contoh proyek GUI yang lebih lengkap: sebuah aplikasi kalkulator sederhana yang menggunakan layout grid dan event handling.

```

import tkinter as tk

class Kalkulator:
    def __init__(self, root):
        self.root = root
        self.root.title('Kalkulator')
        self.root.geometry('320x450')
        self.root.configure(bg='#1e1e1e')
        self.ekspresi = ''
        self.buat_tampilan()
        self.buat_tombol()

```

```

def buat_tampilan(self):
    self.var_display = tk.StringVar(value='0')
    display = tk.Entry(self.root, textvariable=self.var_display,
                        font=('Arial', 28), bd=0, bg='#2d2d2d',
                        fg='white', justify='right', state='readonly')
    display.grid(row=0, column=0, columnspan=4,
                 sticky='nsew', padx=10, pady=10, ipady=15)

def buat_tombol(self):
    tombol_layout = [
        ['C', '±', '%', '÷'],
        ['7', '8', '9', '×'],
        ['4', '5', '6', '-'],
        ['1', '2', '3', '+'],
        ['0', '.', '□', '='],
    ]
    for r, baris in enumerate(tombol_layout, start=1):
        for c, teks in enumerate(baris):
            warna = '#ff9f0a' if teks in ['÷', '×', '-', '+', '='] else '#3a3a3a'
            btn = tk.Button(self.root, text=teks,
                            font=('Arial', 16), bg=warna,
                            fg='white', bd=0, relief='flat',
                            command=lambda t=teks: self.tekan(t))
            btn.grid(row=r, column=c, sticky='nsew',
                    padx=2, pady=2, ipadx=10, ipady=15)

def tekan(self, nilai):
    if nilai == 'C':
        self.ekspresi = ''
        self.var_display.set('0')
    elif nilai == '=':
        try:
            hasil = eval(self.ekspresi.replace('×', '*').replace('÷', '/'))
            self.var_display.set(str(hasil))
            self.ekspresi = str(hasil)
        except: self.var_display.set('Error')
    else:
        self.ekspresi += nilai
        self.var_display.set(self.ekspresi)

root = tk.Tk()
app = Kalkulator(root)
root.mainloop()

```

Contoh 4.3 - Proyek Kalkulator GUI dengan Tkinter

Sumber Referensi — Bab 4

- Roseman, M., & Gries, D. (2019). The Tkinter Reference: A GUI for Python. CreateSpace.

- Moore, A. (2019). Python GUI Programming with Tkinter. Packt Publishing.

- Summerfield, M. (2008). Rapid GUI Programming with Python and Qt. Prentice Hall.

- Dokumentasi resmi Tkinter: <https://docs.python.org/3/library/tkinter.html>

BAB 5: PEMANFAATAN PUSTAKA (LIBRARY)

5.1 Pengertian Pustaka (Library)

Pustaka atau library adalah kumpulan kode program yang sudah ditulis dan diuji sebelumnya, yang dapat digunakan kembali dalam program lain. Library berisi fungsi, kelas, dan modul yang menyediakan fungsionalitas tertentu tanpa perlu ditulis dari awal. Penggunaan library secara efektif adalah salah satu keterampilan terpenting seorang pengembang perangkat lunak profesional.

Menggunakan library yang tepat dapat menghemat waktu pengembangan secara signifikan, mengurangi kemungkinan kesalahan (bug), memanfaatkan kode yang sudah dioptimalkan oleh komunitas, dan memungkinkan pengembang fokus pada logika bisnis aplikasi daripada implementasi detail teknis yang sudah diselesaikan orang lain.

Perbedaan Library, Framework, dan Package

- Library: Koleksi fungsi/kelas yang dipanggil oleh kode Anda. Anda yang mengendalikan alur program.
- Framework: Kerangka kerja yang memanggil kode Anda (inversion of control). Framework mendikte struktur aplikasi.
- Package: Kumpulan modul Python yang terorganisir dalam direktori. Library dan framework biasanya dikemas sebagai package.
- Module: File Python tunggal (.py) yang berisi fungsi, kelas, dan variabel.

5.2 Manajemen Package dengan pip dan venv

pip adalah package manager bawaan Python yang digunakan untuk menginstal, memperbarui, dan menghapus library dari Python Package Index (PyPI). Virtual environment (venv) digunakan untuk membuat lingkungan Python yang terisolasi untuk setiap proyek, mencegah konflik dependensi antar proyek.

```
# Perintah pip dasar
pip install nama_library          # Install library
pip install nama_library==1.2.3  # Install versi spesifik
pip install -r requirements.txt   # Install dari file requirements
pip uninstall nama_library        # Hapus library
```

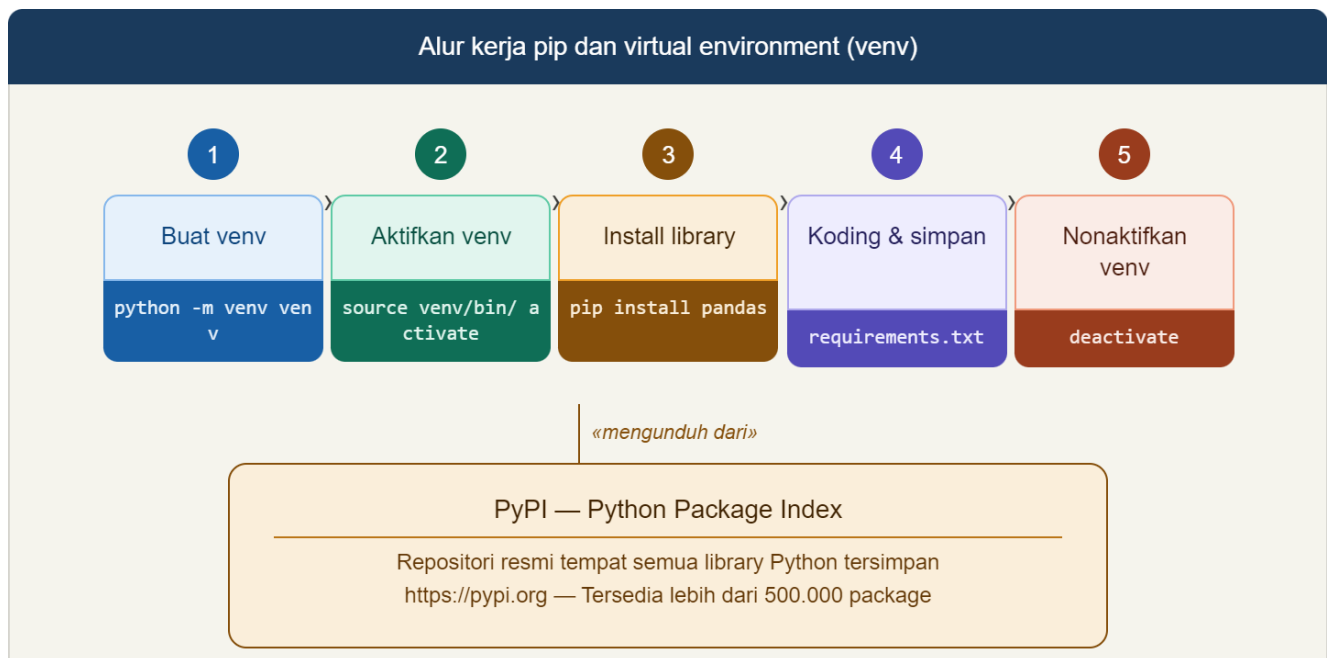
```

pip list                # Tampilkan semua library terinstal
pip freeze > requirements.txt  # Simpan daftar dependensi

# Membuat dan menggunakan virtual environment
python -m venv venv      # Buat virtual environment
venv\Scripts\activate    # Aktifkan (Windows)
source venv/bin/activate  # Aktifkan (Linux/macOS)
deactivate                # Nonaktifkan venv

```

Contoh 5.1 - Perintah pip dan pengelolaan virtual environment



Gambar 5.2 Alur kerja instalasi library menggunakan pip dan virtual environment

5.3 Library Populer Python Berdasarkan Kategori

Kategori	Library	Kegunaan
Manipulasi Data	NumPy	Array multidimensi, operasi matematika numerik
Manipulasi Data	Pandas	DataFrame, analisis dan manipulasi data tabular
Visualisasi Data	Matplotlib	Membuat berbagai jenis grafik dan plot 2D
Visualisasi Data	Seaborn	Visualisasi statistik yang lebih indah berbasis Matplotlib
Machine Learning	Scikit-learn	Algoritma ML klasik: klasifikasi, regresi, clustering

Kategori	Library	Kegunaan
Deep Learning	TensorFlow/Keras	Neural network dan deep learning
Pengembangan Web	Flask	Web framework minimalis untuk API dan web app
Pengembangan Web	Django	Web framework lengkap dengan ORM dan admin panel
Permintaan HTTP	Requests	Mengirim HTTP request ke API dan website
Scraping Web	BeautifulSoup	Parsing HTML dan ekstraksi data dari web
Pengolahan Gambar	Pillow (PIL)	Membuka, memanipulasi, dan menyimpan gambar
Pengolahan Teks	NLTK / spaCy	Natural Language Processing (NLP)
Database	SQLAlchemy	ORM untuk berbagai database (SQLite, MySQL, PostgreSQL)
Pengujian	pytest	Framework pengujian unit dan integrasi

Tabel 5.1 - Library Populer Python Berdasarkan Kategori



Gambar 5.1 Ekosistem library Python populer berdasarkan kategori penggunaan

5.4 NumPy: Komputasi Numerik

NumPy (Numerical Python) adalah library fundamental untuk komputasi saintifik dalam Python. NumPy menyediakan objek array n-dimensi (ndarray) yang efisien, fungsi matematika tingkat tinggi, dan tools untuk integrasi dengan bahasa C, C++, dan Fortran.

```

import numpy as np

# Membuat array NumPy
arr_1d = np.array([1, 2, 3, 4, 5])
arr_2d = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

print('Array 1D:', arr_1d)
print('Array 2D:\n', arr_2d)
print('Shape:', arr_2d.shape)      # (3, 3)
print('Dtype:', arr_1d.dtype)     # int64

# Operasi matematika vektorisasi
a = np.array([1, 2, 3, 4])
b = np.array([5, 6, 7, 8])
print('Penjumlahan:', a + b)      # [6, 8, 10, 12]
print('Perkalian:', a * b)       # [5, 12, 21, 32]
print('Dot product:', np.dot(a, b)) # 70

# Fungsi statistik

```

```
data = np.array([23, 45, 12, 67, 34, 89, 56])
print(f'Mean: {np.mean(data):.2f}')
print(f'Median: {np.median(data):.2f}')
print(f'Std Dev: {np.std(data):.2f}')
print(f'Min: {np.min(data)}, Max: {np.max(data)}')
```

Contoh 5.2 - Penggunaan Dasar NumPy

5.5 Pandas: Analisis Data

Pandas adalah library Python yang menyediakan struktur data dan alat analisis data yang cepat, fleksibel, dan mudah digunakan. Dua struktur data utama Pandas adalah Series (data satu dimensi) dan DataFrame (data dua dimensi seperti tabel). Pandas sangat berguna untuk membaca, membersihkan, dan menganalisis data dari berbagai sumber seperti CSV, Excel, SQL, dan JSON.

```
import pandas as pd

# Membuat DataFrame
data_siswa = {
    'Nama'      : ['Budi', 'Ani', 'Candra', 'Dina', 'Eka'],
    'Nilai_MTK' : [85, 92, 78, 88, 76],
    'Nilai_IPA' : [80, 88, 82, 90, 72],
    'Nilai_B.ING': [78, 95, 70, 85, 80],
}
df = pd.DataFrame(data_siswa)

# Menambahkan kolom rata-rata
df['Rata_Rata'] = df[['Nilai_MTK', 'Nilai_IPA', 'Nilai_B.ING']].mean(axis=1)
df['Status'] = df['Rata_Rata'].apply(lambda x: 'Lulus' if x >= 75 else 'Tidak Lulus')

print(df.to_string(index=False))
print('\nStatistik:')
print(df[['Nilai_MTK', 'Nilai_IPA']].describe())

# Filtering data
siswa_lulus = df[df['Status'] == 'Lulus']
print(f'\nJumlah siswa lulus: {len(siswa_lulus)}')

# Menyimpan ke CSV
df.to_csv('data_siswa.csv', index=False)
```

Contoh 5.3 - Penggunaan Pandas untuk Analisis Data Siswa

5.6 Matplotlib: Visualisasi Data

Matplotlib adalah library visualisasi data 2D yang paling banyak digunakan dalam Python. Matplotlib memungkinkan pembuatan berbagai jenis grafik: line chart, bar chart, scatter plot, histogram, pie chart, dan banyak lagi. Library ini sangat berguna untuk eksplorasi data dan presentasi hasil analisis.


```

import matplotlib.pyplot as plt
import numpy as np

# Membuat figure dengan beberapa subplot
fig, axes = plt.subplots(1, 2, figsize=(12, 5))
fig.suptitle('Contoh Visualisasi Data dengan Matplotlib', fontsize=14)

# Subplot 1: Bar Chart
mata_pelajaran = ['Matematika', 'IPA', 'B. Inggris', 'Sejarah']
rata_rata = [82, 88, 75, 79]
warna = ['#2E75B6', '#27AE60', '#E67E22', '#8E44AD']
axes[0].bar(mata_pelajaran, rata_rata, color=warna, edgecolor='white')
axes[0].set_title('Rata-rata Nilai per Mata Pelajaran')
axes[0].set_ylabel('Nilai')
axes[0].set_ylim(60, 100)
for i, v in enumerate(rata_rata):
    axes[0].text(i, v + 0.5, str(v), ha='center', fontweight='bold')

# Subplot 2: Line Chart
bulan = ['Jan', 'Feb', 'Mar', 'Apr', 'Mei', 'Jun']
nilai_a = [80, 85, 78, 90, 88, 92]
nilai_b = [72, 75, 80, 77, 83, 86]
axes[1].plot(bulan, nilai_a, 'bo-', label='Kelas A', linewidth=2)
axes[1].plot(bulan, nilai_b, 'rs--', label='Kelas B', linewidth=2)
axes[1].set_title('Perkembangan Nilai per Bulan')
axes[1].legend()
axes[1].grid(True, alpha=0.3)

plt.tight_layout()
plt.savefig('grafik_nilai.png', dpi=150, bbox_inches='tight')
plt.show()

```

Contoh 5.4 - Visualisasi Data dengan Matplotlib

5.7 Requests: Mengakses API Web

Library Requests adalah cara paling mudah untuk mengirim permintaan HTTP di Python. Library ini memungkinkan pengambilan data dari API web, yang merupakan salah satu kebutuhan paling umum dalam pengembangan aplikasi modern. Dengan Requests, kita dapat mengambil data dari internet, mengirimkan data, dan bekerja dengan berbagai format seperti JSON, XML, dan HTML.

```

import requests
import json

# Mengakses API publik (contoh: JSONPlaceholder)
url = 'https://jsonplaceholder.typicode.com/posts'

# GET Request - mengambil data
response = requests.get(url, params={'_limit': 5})

```

```
# Cek status response
if response.status_code == 200:
    data = response.json() # Parse JSON otomatis
    print(f'Berhasil! Mendapat {len(data)} post')
    for post in data[:3]:
        print(f"ID: {post['id']} | Judul: {post['title'][:40]}...")
else:
    print(f'Gagal! Status code: {response.status_code}')

# POST Request - mengirim data
data_baru = {
    'title': 'Belajar Python',
    'body': 'Python sangat menyenangkan!',
    'userId': 1
}
post_response = requests.post(url, json=data_baru)
print(f'POST Status: {post_response.status_code}')
print(f'Data terkirim ID: {post_response.json()["id"]}')

```

Contoh 5.5 - Mengakses API Web dengan Library Requests

5.8 Studi Kasus: Proyek Analisis Data Penjualan

Berikut adalah contoh proyek nyata yang menggabungkan beberapa library sekaligus: NumPy untuk komputasi, Pandas untuk manipulasi data, dan Matplotlib untuk visualisasi. Proyek ini menganalisis data penjualan toko dan menghasilkan laporan visual.

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

# Simulasi data penjualan (biasanya dibaca dari file CSV)
np.random.seed(42)
bulan = ['Jan', 'Feb', 'Mar', 'Apr', 'Mei', 'Jun',
          'Jul', 'Agu', 'Sep', 'Okt', 'Nov', 'Des']
data = {
    'Bulan' : bulan,
    'Elektronik': np.random.randint(50_000_000, 200_000_000, 12),
    'Fashion' : np.random.randint(30_000_000, 120_000_000, 12),
    'Makanan' : np.random.randint(20_000_000, 80_000_000, 12),
}
df = pd.DataFrame(data)
df['Total'] = df[['Elektronik', 'Fashion', 'Makanan']].sum(axis=1)

# Analisis
print('=== RINGKASAN PENJUALAN TAHUNAN ===')
print(f'Total Penjualan: Rp{df["Total"].sum():,.0f}')
print(f'Bulan Terbaik: {df.loc[df["Total"].idxmax(), "Bulan"]}')
print(f'Rata-rata per Bulan: Rp{df["Total"].mean():,.0f}')

```

```
# Visualisasi
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 5))

# Stacked bar chart
df.plot(x='Bulan', y=['Elektronik','Fashion','Makanan'],
        kind='bar', stacked=True, ax=ax1, colormap='Blues')
ax1.set_title('Penjualan per Kategori per Bulan')
ax1.set_ylabel('Penjualan (Rupiah)')

# Pie chart total per kategori
kategori_total = df[['Elektronik','Fashion','Makanan']].sum()
ax2.pie(kategori_total, labels=kategori_total.index,
        autopct='%1.1f%%', colors=['#2E75B6','#E67E22','#27AE60'])
ax2.set_title('Proporsi Penjualan per Kategori')

plt.tight_layout()
plt.savefig('laporan_penjualan.png', dpi=150)
print('Laporan visual tersimpan sebagai laporan_penjualan.png')
```

Contoh 5.6 - Proyek Analisis Data Penjualan menggunakan NumPy, Pandas, dan Matplotlib

Sumber Referensi — Bab 5

- McKinney, W. (2022). Python for Data Analysis: Data Wrangling with pandas, NumPy & Jupyter (3rd ed.). O'Reilly Media.
- VanderPlas, J. (2023). Python Data Science Handbook (2nd ed.). O'Reilly Media.
- Harris, C. R., et al. (2020). Array programming with NumPy. Nature, 585, 357-362.
- Dokumentasi resmi NumPy: <https://numpy.org/doc/> | Pandas: <https://pandas.pydata.org/docs/>

RANGKUMAN

Materi pembelajaran ini mencakup lima topik utama yang membentuk kompetensi dasar seorang pengembang perangkat lunak modern. Berikut adalah poin-poin kunci dari setiap topik:

ATP 2.1 — Pemrograman Terstruktur

- Pemrograman terstruktur menggunakan tiga struktur kendali utama: Sequence, Selection, dan Repetition untuk mengorganisasikan kode program.
- Prinsip-prinsip penting meliputi modularisasi (memecah program menjadi fungsi-fungsi kecil), abstraksi, dan stepwise refinement.
- Penggunaan fungsi/prosedur membuat kode lebih mudah dipahami, diuji, dan digunakan kembali (reusable).

ATP 2.2 — Pemrograman Berorientasi Objek Lanjutan

- Empat pilar OOP: Enkapsulasi (penyembunyian data), Pewarisan (inheritance), Polimorfisme (banyak bentuk), dan Abstraksi.
- Abstract Base Class (ABC) memungkinkan pembuatan template class yang wajib diimplementasikan oleh subclass.
- Design Patterns seperti Singleton, Factory, dan Observer adalah solusi standar untuk masalah desain yang sering muncul.

ATP 2.3 — Pemodelan Perangkat Lunak

- UML (Unified Modeling Language) adalah standar industri untuk memodelkan perangkat lunak menggunakan berbagai jenis diagram.
- Use Case Diagram menggambarkan interaksi aktor dengan sistem; Class Diagram menggambarkan struktur statis; Sequence Diagram menggambarkan alur interaksi.
- Pemodelan membantu komunikasi antar tim, dokumentasi sistem, dan deteksi masalah desain lebih awal.

ATP 2.4 — Pemrograman Antarmuka GUI

- GUI programming memungkinkan pembuatan aplikasi desktop dengan antarmuka grafis yang user-friendly.
 - Tkinter adalah framework GUI bawaan Python yang cocok untuk pembelajaran, dengan widget, layout manager (pack/grid/place), dan event handling.
 - Konsep penting: Widget, Container, Layout Manager, Event, Event Handler, dan Event Loop.
-

ATP 2.5 — Pemanfaatan Pustaka (Library)

- Library memungkinkan pengembang menggunakan kode yang sudah teruji tanpa menulis dari awal, menghemat waktu dan mengurangi bug.
- pip digunakan untuk manajemen package; virtual environment (venv) untuk isolasi dependensi proyek.
- Library penting: NumPy (komputasi numerik), Pandas (analisis data), Matplotlib (visualisasi), Requests (HTTP/API), dan banyak lagi.

DAFTAR PUSTAKA

- Deitel, P., & Deitel, H. (2021). Python How to Program (3rd ed.). Pearson Education.
- Dijkstra, E. W. (1968). Go To Statement Considered Harmful. Communications of the ACM, 11(3), 147-148.
- Fowler, M. (2003). UML Distilled: A Brief Guide to the Standard Object Modeling Language (3rd ed.). Addison-Wesley.
- Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley.
- Harris, C. R., et al. (2020). Array programming with NumPy. Nature, 585, 357-362.
- Hetland, M. L. (2017). Python Algorithms: Mastering Basic Algorithms in the Python Language (2nd ed.). Apress.
- Lutz, M. (2019). Learning Python (5th ed.). O'Reilly Media.
- Martin, R. C. (2008). Clean Code: A Handbook of Agile Software Craftsmanship. Prentice Hall.
- McKinney, W. (2022). Python for Data Analysis: Data Wrangling with pandas, NumPy & Jupyter (3rd ed.). O'Reilly Media.
- Moore, A. (2019). Python GUI Programming with Tkinter. Packt Publishing.
- Phillips, D. (2018). Python 3 Object-Oriented Programming (3rd ed.). Packt Publishing.
- Pressman, R. S., & Maxim, B. R. (2020). Software Engineering: A Practitioner's Approach (9th ed.). McGraw-Hill.
- Rumbaugh, J., Jacobson, I., & Booch, G. (2005). The Unified Modeling Language Reference Manual (2nd ed.). Addison-Wesley.
- Sebesta, R. W. (2019). Concepts of Programming Languages (12th ed.). Pearson Education.
- Sommerville, I. (2016). Software Engineering (10th ed.). Pearson Education.
- VanderPlas, J. (2023). Python Data Science Handbook (2nd ed.). O'Reilly Media.

Sumber Online:

- Dokumentasi Python resmi: <https://docs.python.org>
 - Dokumentasi NumPy: <https://numpy.org/doc/>
 - Dokumentasi Pandas: <https://pandas.pydata.org/docs/>
 - Dokumentasi Matplotlib: <https://matplotlib.org/stable/contents.html>
-

- Dokumentasi Tkinter: <https://docs.python.org/3/library/tkinter.html>
- PyPI (Python Package Index): <https://pypi.org>