KUSUMA WEB | EDU SERIES

MODUL 10: PEMROGRAMAN BERORIENTASI OBJEK

MASTERCLASS PYTHON DASAR

Memahami Paradigma Arsitektur Pemrograman OOP Skala Industri. Eksplorasi Konsep `Class` & `Object`, Konstruktor Khusus `__init__`, Atribut & Perilaku (*Method*), serta Konsep Pewarisan Sifat (*Inheritance*).

KUSUMA WEB

Kusuma Web Edu Series
Kelas Dasar Pemrograman — Modul 10 dari 10
Tanggal Rilis: 29 Juni 2026

Hak Cipta © 2026 Kusuma Web. All Rights Reserved.

Memahami Paradigma OOP, `Class` & `Object`

Object-Oriented Programming (OOP) atau Pemrograman Berorientasi Objek adalah paradigma pemrograman terpopuler di dunia industri saat ini. Konsep dasar OOP adalah menyusun kode program dengan cara memodelkan objek-objek dunia nyata secara terstruktur ke dalam program komputer.

1. Apa itu `Class` (Kelas)?

`Class` bertindak sebagai cetak biru (*blueprint*) atau cetakan dasar yang mendefinisikan struktur data, sifat, dan perilaku awal dari suatu entitas sebelum diwujudkan menjadi nyata.

2. Apa itu `Object` (Objek)?

`Object` adalah wujud fisik nyata dari sebuah kelas (*instance of class*). Kita dapat membuat objek yang tak terbatas jumlahnya menggunakan satu cetakan kelas yang sama, masing-masing dengan nilai data yang unik.

- **Atribut (Data):** Variabel di dalam objek yang menyimpan informasi kondisi fisik (contoh: merk, warna, roda).
- **Method (Perilaku):** Fungsi di dalam objek yang melakukan aksi tindakan fisik (contoh: berjalan, mengerem).

Representasi Sederhana

Sama seperti cetakan kue (*Class*), kita dapat menciptakan ribuan kue nyata (*Object*) dengan rasa, dekorasi, dan isian krim yang berbeda-beda (*Atribut*).

KUSUMA WEB

Konstruktor `__init__` & Peran Parameter `self`

Untuk merealisasikan objek secara cerdas, Python menyediakan struktur metode khusus yang bertugas mempersiapkan alokasi awal atribut objek saat diciptakan.

1. Metode Konstruktor `__init__`

Konstruktor adalah metode khusus bawaan Python yang akan **otomatis dijalankan sekali saja** oleh sistem begitu kita membuat objek nyata dari suatu kelas. Digunakan untuk mendefinisikan nilai atribut awal objek.

2. Peran Parameter Utama `self`

Parameter `self` adalah referensi internal yang mewakili objek spesifik yang sedang aktif bekerja saat ini. Gunakan `self` untuk membedakan variabel internal kelas dari variabel umum di luarnya.

3. Contoh Implementasi Kelas Kendaraan

```
class Mobil:
    # Metode Konstruktor inisialisasi awal
    def __init__(self, merk, warna):
        self.merk = merk      # Atribut
        self.warna = warna    # Atribut

    # Metode Perilaku (Method)
    def klakson(self):
        print(f"Mobil {self.merk} berwarna {self.warna} berbunyi: Tiiit Tiiit!
            ")

# Instansiasi membuat objek nyata
mobil_rian = Mobil("Toyota Supra", "Merah")
mobil_rian.klakson() # Memanggil method
```

Pilar Utama OOP: Pewarisan (*Inheritance*)

Salah satu keunggulan terbesar OOP adalah kemampuannya dalam memperluas jangkauan kode tanpa perlu menulis ulang cetakan logika dari nol, yang diwujudkan melalui konsep **Pewarisan (*Inheritance*)**.

1. Apa itu *Inheritance*?

Konsep pewarisan memungkinkan suatu kelas baru (*Child Class / Subclass*) mewarisi seluruh atribut dan metode dari kelas yang sudah ada sebelumnya (*Parent Class / Superclass*) secara otomatis.

2. Mengapa Ini Penting?

Mendukung penuh modularitas kode pemrograman, mengurangi redundansi penulisan, serta mempermudah pengembangan fitur-fitur baru ke depannya.

3. Contoh Kode Pewarisan Sifat Kelas

```python
# Kelas Induk (Parent Class)
class Hewan:
    def __init__(self, nama):
        self.nama = nama
    def bersuara(self):
        pass # Akan didefinisikan ulang oleh anak kelas

# Kelas Anak mewarisi sifat Hewan
class Kucing(Hewan):
    def bersuara(self):
        return "Meow! Meow!"

kucing_lucu = Kucing("Kitty")
print(f"Nama Hewan: {kucing_lucu.nama} | Bunyi: {kucing_lucu.bersuara()}")
```

Studi Kasus: Sistem Simulasi Akun ATM Digital

Mari kita bangun sebuah rancangan arsitektur akun bank digital Kusuma Web yang tangguh memanfaatkan implementasi pemrograman berorientasi objek (OOP) secara penuh.

1. Logika Akun Bank

Setiap objek rekening nasabah (*Bank Account*) memiliki saldo pribadi yang hanya bisa dimanipulasi melalui metode aman `deposit()` untuk menabung, dan `withdraw()` untuk penarikan dana bersyarat.

2. Kode Lengkap Program

```
# Simulasi Rekening Bank Berbasis OOP
class RekeningBank:
    def __init__(self, nama_pemilik, saldo_awal):
        self.pemilik = nama_pemilik
        self.saldo = saldo_awal # Atribut saldo tersimpan

    # Method menabung uang
    def deposit(self, jumlah):
        self.saldo += jumlah
        print(f"[SETOR] Berhasil menyetor Rp {jumlah}. Saldo baru: Rp {self.
            saldo}")

    # Method menarik uang dengan validasi kecukupan dana
    def withdraw(self, jumlah):
        if jumlah <= self.saldo:
            self.saldo -= jumlah
            print(f"[TARIK] Berhasil menarik Rp {jumlah}. Saldo sisa: Rp {self
                .saldo}")
        else:
            print(f"[GAGAL] Penarikan Rp {jumlah} ditolak! Saldo Anda tidak
                cukup.")

# --- Demo Operasi Objek Nyata ---
print("=== SISTEM DIGITAL ATM KUSUMA WEB ===")
akun_budi = RekeningBank("Budi Santoso", 500000)

akun_budi.deposit(150000)
akun_budi.withdraw(100000)
akun_budi.withdraw(600000) # Memicu penarikan gagal
print("=====================================")
```