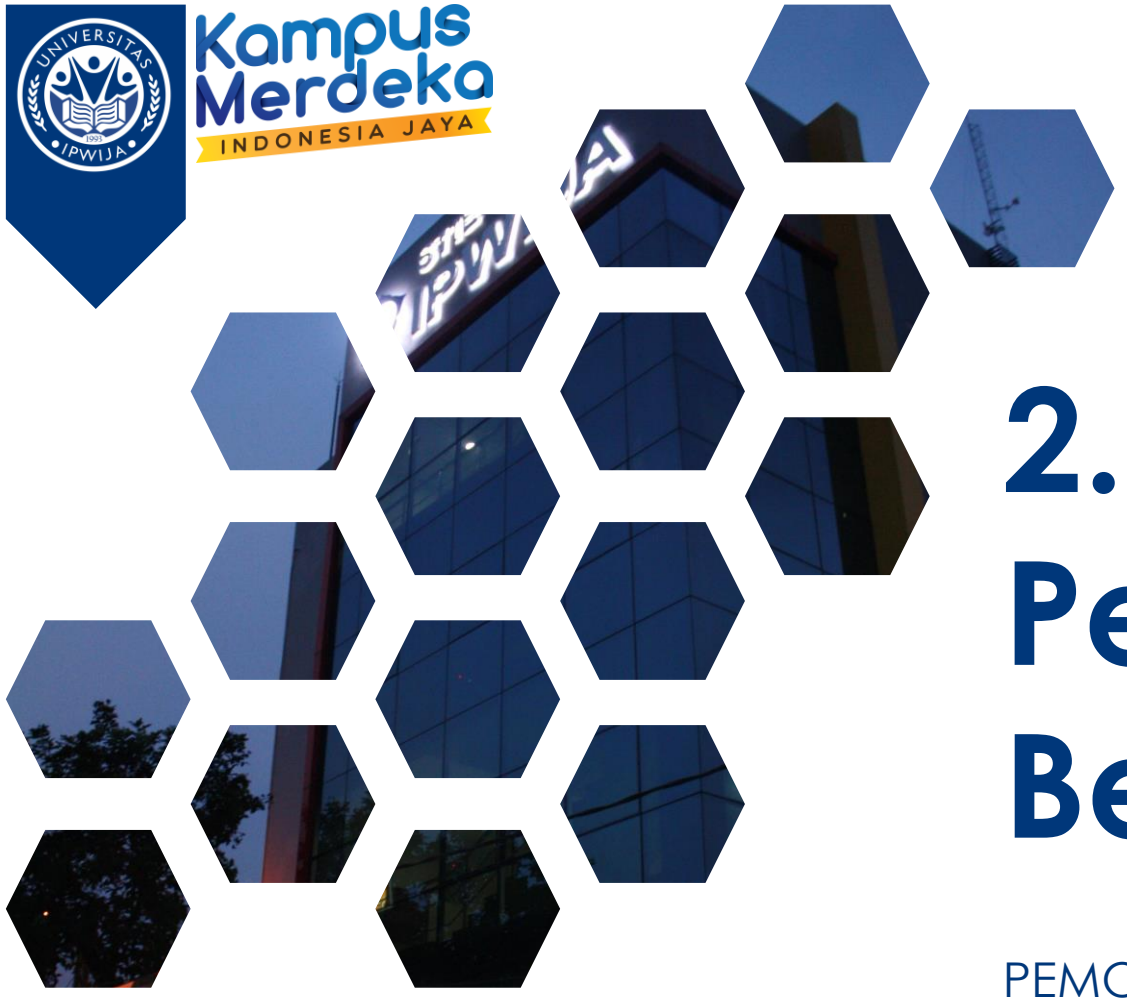
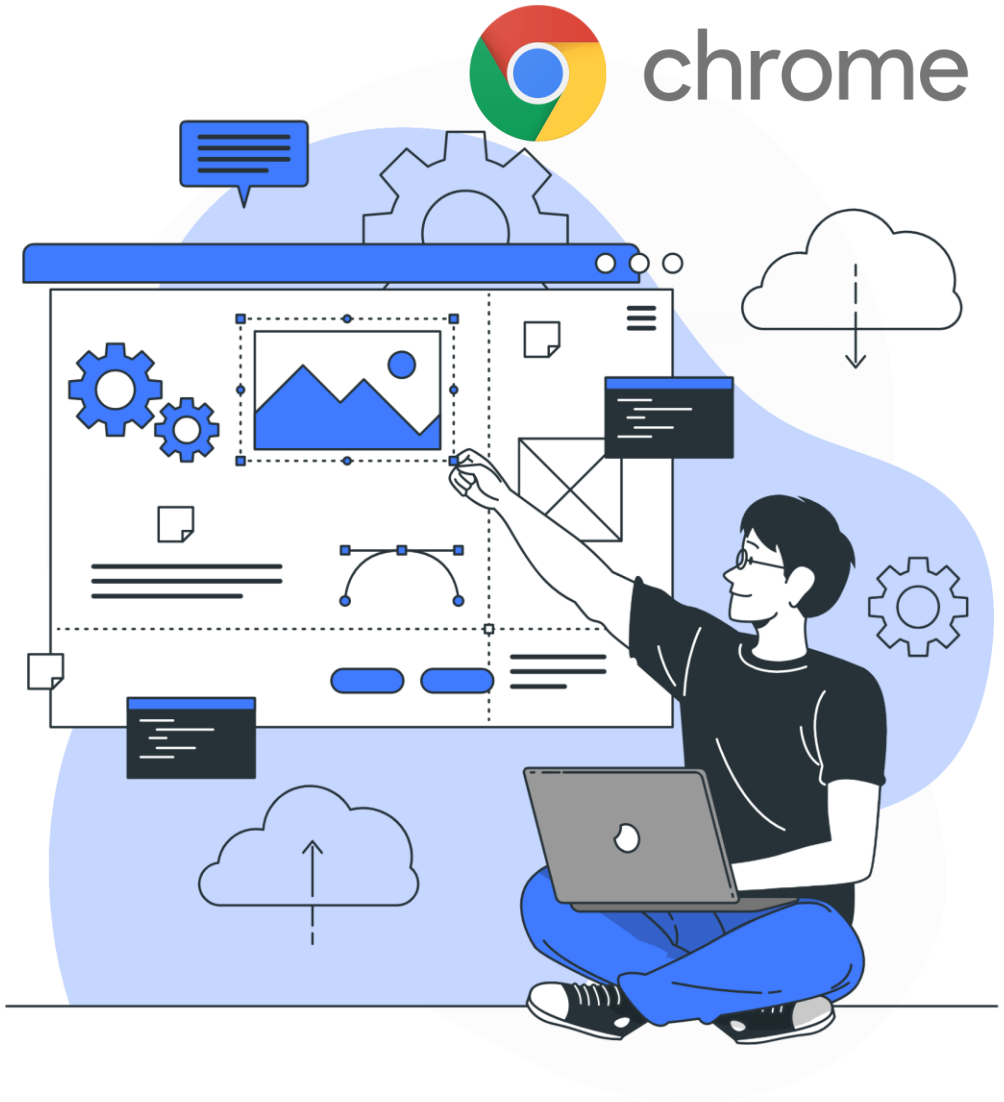


2. Konsep Dasar Pemrograman Berorientasi Objek

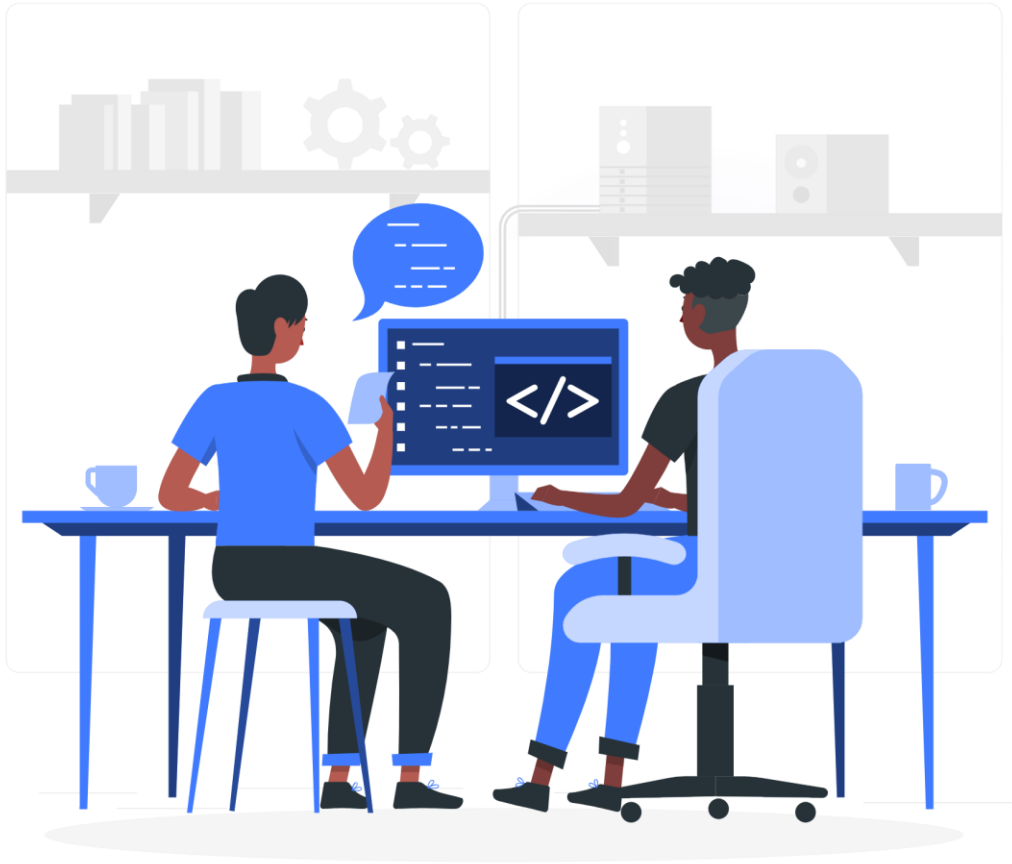
PEMOGRAMAN WEB

Paradigma pemrograman (Prosedural vs. OOP)



Prinsip dasar OOP: Encapsulation, Inheritance, Polymorphism, Abstraction

- Encapsulation,
- Inheritance,
- Polymorphism,
- Abstraction



Encapsulation

Mengemas data dan fungsi dalam satu kesatuan agar lebih aman dan modular

```
class BankAccount {
    #saldo;  // Properti private
    constructor(nama, saldoAwal) {
        this.nama = nama;
        this.#saldo = saldoAwal;
    }
    getSaldo() {
        return `Saldo ${this.nama} adalah Rp${this.#saldo}`;
    }
    deposit(jumlah) {
        this.#saldo += jumlah;
    }
}

const akun = new BankAccount("Alice", 500000);
console.log(akun.getSaldo());
akun.deposit(200000);
console.log(akun.getSaldo());
// console.log(akun.#saldo); // Akan error karena saldo bersifat private
```

Inheritance

Memungkinkan objek mewarisi karakteristik dari objek lain, mengurangi redundansi kode.

```
class Hewan {
  constructor(nama) {
    this.nama = nama;
  }
  bersuara() {
    return `${this.nama} bersuara...`;
  }
}
class Kucing extends Hewan {
  constructor(nama, warna) {
    super(nama);
    this.warna = warna;
  }
  bersuara() {
    return `${this.nama} mengeong!`;
  }
}
const kucing = new Kucing("Milo", "Hitam");
console.log(kucing.bersuara()); // Milo mengeong!
```

dalam contoh pewarisan di atas, metode bersuara() punya hasil berbeda di Hewan dan Kucing

Polymorphism

Memungkinkan satu antarmuka memiliki berbagai implementasi, meningkatkan fleksibilitas.

Abstraction

Menyederhanakan kompleksitas dengan hanya menampilkan fitur penting.

```
class Kendaraan {
    constructor(nama) {
        if (this.constructor === Kendaraan) {
            throw new Error("Kelas abstrak tidak bisa diinstansiasi!");
        }
        this.nama = nama;
    }
    bergerak() {
        throw new Error("Metode ini harus diimplementasikan di subclass!");
    }
}
class Motor extends Kendaraan {
    bergerak() {
        return `${this.nama} berjalan di jalan raya!`;
    }
}
const supra = new Motor("Supra X");
console.log(supra.bergerak());
```

Studi kasus sederhana

Dengan OOP, kode ini menjadi lebih modular, mudah diperluas, dan lebih terstruktur dibandingkan pendekatan prosedural.

```
class Buku {
    constructor(judul, penulis) {this.judul = judul; this.penulis = penulis; }
    info() { return `Buku: ${this.judul}, ditulis oleh ${this.penulis}.`; }
}

class Perpustakaan {
constructor() { this.koleksi = [];}
tambahBuku(buku) { this.koleksi.push(buku); }
tampilkanBuku() { this.koleksi.forEach(b => console.log(b.info())); }
}

const buku1 = new Buku("Harry Potter", "J.K. Rowling");
const perpustakaan = new Perpustakaan();
perpustakaan.tambahBuku(buku1);
perpustakaan.tampilkanBuku();
```

Praktikum:

- Tulis kode di github