# Slide 1: Introduktion til OOP

God dag alle sammen. I dag vil vi dykke ned i verdenen af Objektorienteret Programmering, eller OOP, i Python. Denne tilgang er ikke blot en programmeringsstil, men en paradigme revolution. Python's OOP er enkel, intuitiv og utrolig kraftfuld. Vi fokuserer på 'objekter' - samlinger af data og metoder - i stedet for traditionelle sekventielle instruktioner. Dette giver os en mere naturlig måde at tænke på softwarekonstruktion, hvor vi modellerer virkelige objekter eller koncepter med deres egenskaber og adfærd.

# Slide 2: OOP Grundlæggende

OOP i Python er bygget op omkring 'klasser', som er som skabeloner for at skabe ensartede 'objekter'. Tænk på klasser som blåprints, der definerer, hvordan et objekt ser ud og opfører sig. Hvert objekt er en konkret instans af en klasse, med sine egne unikke egenskaber. En vigtig del af OOP er 'arv', som giver os mulighed for at udvide og tilpasse eksisterende kode. Arv tillader nye klasser at overtage eller 'arve' egenskaber og metoder fra eksisterende klasser, hvilket fremmer genbrugbarhed og modularitet i vores kode.

# Slide 3: Klasse 'Person' - Struktur

Se på vores eksempel her, klassen 'Person'. Det er en simpel model, men den demonstrerer kraften i OOP. Vi definerer grundlæggende egenskaber som navn og alder. Ved at bruge private attributter, som indikeret af understregningen, sikrer vi, at disse data kun kan tilgås og ændres gennem vores klassemetoder. Dette beskytter vores data mod uautoriseret adgang og manipulation.

# Slide 4: Klasse 'Person' - Implementering

I denne del af klassen 'Person' ser vi, hvordan vi opretter en person ved hjælp af konstruktøren \_\_init\_\_. Denne metode kaldes, når vi opretter en ny instans af vores klasse. Den initialiserer de private attributter \_navn og \_alder, hvilket er afgørende for at sikre, at vores objekter oprettes med de korrekte data fra starten.

# Slide 5: Forståelse af @property

Her introducerer vi @property-dekorationen. Det er en smart måde at håndtere datahåndtering på i Python. Med @property kan vi definere en metode, som agerer som en 'getter' for en attribut. Det betyder, at når vi vil tilgå en privat attribut som navn, skal vi gøre det gennem denne metode, som sikrer, at adgangen sker på en kontrolleret og sikker måde.

# Slide 6: Implementering af Getter og Setter

På denne slide ser vi på setter-metoden for 'navn'. Denne del af koden tillader os at opdatere værdien af 'navn'. Setter-metoden er udstyret med valideringslogik, hvilket betyder, at vi kan kontrollere, om inputtet er gyldigt, før vi lader det ændre den underliggende data. Det er en nøglekomponent i indkapsling, da det hjælper os med at bevare integriteten af vores data.

# Slide 7: Anvendelse af Klasse 'Person'

Nu kommer vi til anvendelsen af vores klasse. Vi opretter et nyt 'Person'-objekt med navnet 'Lars' og alderen 30. Når vi kalder metoden vis\_info(), ser vi, hvordan klassen tillader os at interagere med dens data på en sikker måde. Dette viser, hvordan OOP gør det nemt at arbejde med komplekse datastrukturer på en intuitiv måde.

# Slide 8: Opdatering og Fejlhåndtering

Her demonstreres opdatering af objektegenskaber og vigtigheden af fejlhåndtering. Vi ser, hvordan vi kan ændre 'Lars' navn til 'Mette' og opdatere alderen til 35. Men hvad sker der, når vi prøver at sætte en ugyldig værdi? Vores klasse håndterer dette elegant ved at kaste en fejl, hvilket forhindrer korruption af vores objekts tilstand.

# Slide 9: Sammenfatning af OOP og Indkapsling

OOP er en af hjørnestenene i moderne softwareudvikling, især i Python. Ved at bruge koncepter som klasser og indkapsling, kan vi opbygge robuste, genanvendelige og vedligeholdelsesvenlige systemer. Indkapsling, som vi har set med @property, forbedrer vores programstruktur og sikrer datasikkerhed, hvilket er afgørende i et sikkert og effektivt softwaremiljø.

# Slide 10: Afsluttende Bemærkninger

Til sidst, objektorienteret programmering i Python er ikke bare en kodningsteknik, det er en stærk værktøjskasse, der åbner op for en verden af muligheder. Det fører til mere robust, fleksibel og vedligeholdelsesvenlig kode. Tak for jeres opmærksomhed. Er der nogle spørgsmål?