Gruppetime uke 11
IN1000

Plan for timen

- Hvordan lage gode modulære programmer med OOP
- Kort repetisjon av regler for innkapsling
- Tidligere eksamensoppgaver
- Studentsystem oppgave

Hvordan lage gode modulære programmer med OOP

Hva innebærer det å leg et program som er modulært?

- Organisere koden i små uavhengige moduler
- Hver komponent har ansvar for å løse en spesifikk oppgave
  - En metode skal for eksempel bare utføre en oppgave
- Moduler kan tenkes på som byggesteiner i programmet
- I sum handler det om å følge prinsippet om "enhetlig ansvar" og å bryte opp store og komplekse problemer i mindre, mer håndterbare biter, som hver har sitt eget klart definerte formål. Dette gjør det lettere å bygge og vedlikeholde kode.

Husk så mange av tallene du kan på neste slide på 5 sekunder

7 2 9 5 4 7 2 9 4 5 7 8

Del opp problemet i biter

- Det å huske alle tallene på en gang kan virke utfordrende og forvirrende, akkurat som et stort og rotete program kan være vanskelig å forstå
- For å gjøre dette lettere kan vi kanskje dele opp tallrekken i seksjoner på 4 tall hver
  - Blir som å huske tre årstall

7295 4729 4578

Fordeler med å dele opp programmet i mindre, enklere problemer

Fordeler med å dele opp programmet i mindre, enklere problemer

- Akkurat som å dele opp tallrekken gjør det enklere å håndtere tallene, deler modulær programmering opp store og komplekse programmer i mindre, selvstendige moduler, hver med et bestemt ansvar.
- Vi skal nå se på noen fordeler med å dele opp kode på denne måten, herunder:

1. Lesbarhet og vedlikehold
2. Gjenbruk av kode
3. Abstraksjon
4. Innkapsling

Lesbarhet og vedlikehold

- Å dele opp koden i klasser og metoder gjør den mer lesbar og forståelig ved å bryte den ned i mindre deler med klart definerte oppgaver. Dette gjør det lettere å forstå, vedlikeholde og utvide programmet over tid.
- Det er ofte lurt å kalle metodene etter oppgaven de skal utføre. Gi gjerne også parameterne korte og forståelige navn, for eksempel:
  - adder(tall1, tall2)
  - finn_maks_verdi(tall_liste)
  - gjor_til_store_bokstaver(streng)

Eksempel

Klasse med én metode som gjør mer enn én oppgave:

```
class EnkelKalkulator:
    def beregn(self, tall1, tall2, operasjon):
        if operasjon == 'addisjon':
            resultat = tall1 + tall2
        elif operasjon == 'subtraksjon':
            resultat = tall1 - tall2
        elif operasjon == 'multiplikasjon':
            resultat = tall1 * tall2
        elif operasjon == 'divisjon':
            resultat = tall1 / tall2
        else:
            resultat = 'Ugyldig operasjon'
        print(f'Resultatet er: {resultat}')
```

Klasse med flere metoder som utfører en enkelt oppgave hver:

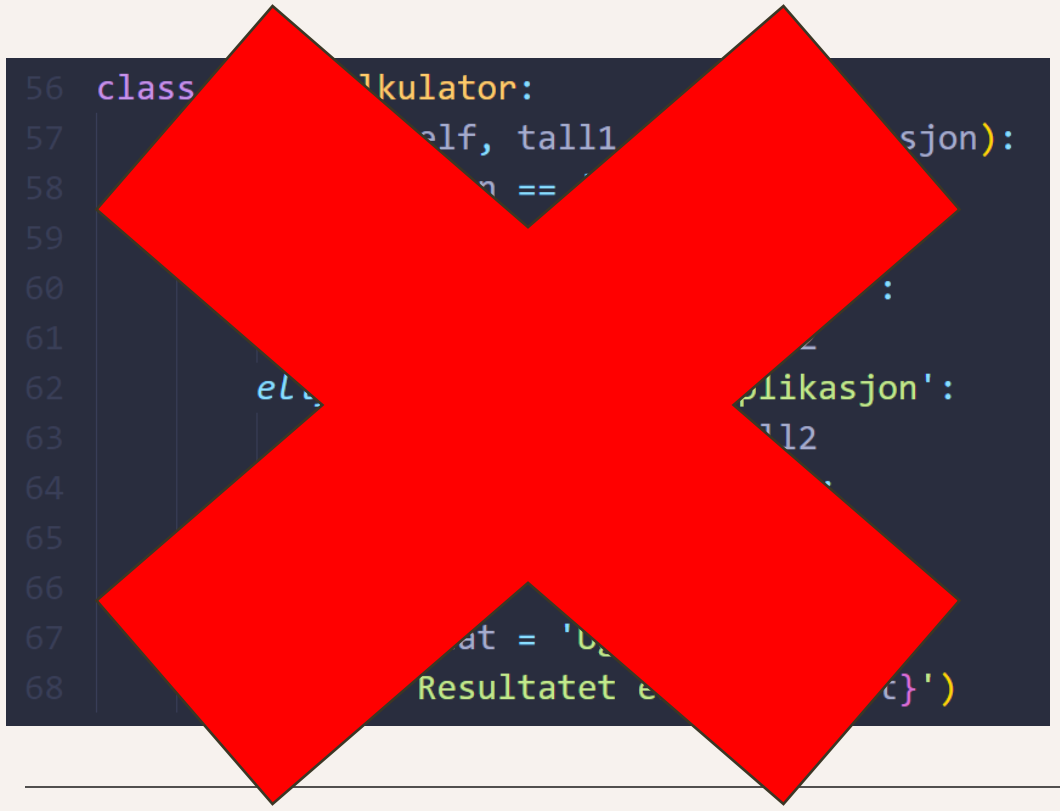
```
class ForbedretKalkulator:
    def addisjon(self, tall1, tall2):
        return tall1 + tall2

    def subtraksjon(self, tall1, tall2):
        return tall1 - tall2

    def multiplikasjon(self, tall1, tall2):
        return tall1 * tall2

    def divisjon(self, tall1, tall2):
        if tall2 == 0:
            return 'Kan ikke dele på null'
        return tall1 / tall2
```

Eksempel



```
56 class
57
58
59
60
61
62
63
64
65
66
67
68
```

```python
class ForbedretKalkulator:
    def addisjon(self, tall1, tall2):
        return tall1 + tall2

    def subtraksjon(self, tall1, tall2):
        return tall1 - tall2

    def multiplikasjon(self, tall1, tall2):
        return tall1 * tall2

    def divisjon(self, tall1, tall2):
        if tall2 == 0:
            return 'Kan ikke dele på null'
        return tall1 / tall2
```

Gjenbrukbarhet

- Når du har modulert koden, kan du enklere gjenbruke modulene i forskjellige deler av programmet eller til og med i andre prosjekter. Dette sparer tid og reduserer sjansene for feil.
- Her kan man bruke sum_tall() og tell_tall() metodene istedenfor å skrive koden for å summere tall og telle tall på nytt

```
    def bergen_gjennomsnitt(self, tall):
        # Bruk to gjenbruksmetoder for å beregne gjennomsnittet
        total = self.sum_tall(tall)
        teller = self.tell_tall(tall)

        # Beregn gjennomsnittet ved å dele totalen på antallet tall
        if teller > 0:
            snitt = total / teller
            return snitt
        else:
            return 0

    def sum_tall(self, tall):
        # Metode for å beregne summen av tallene i listen
        total = sum(tall)
        return total

    def tell_tall(self, tall):
        # Metode for å telle antallet tall i listen
        teller = len(tall)
        return teller
```

Abstraksjon

- Moduler gir en abstraksjon som lar deg fokusere på hvordan hver del fungerer, uten å bekymre deg for detaljer på et høyere nivå.
- Det å abstrahere vekk detaljene til koden bidrar til bedre lesbarhet, som gjør det lettere å vedlikeholde programmet

```
47  num_list = [10, 20, 30, 40, 50]
48  result = bergen_gjennomsnitt(num_list)
49  print("Gjennomsnitt:", result)
```

```
26  # Uten abstraksjon
27  num_list = [10, 20, 30, 40, 50]
28
29  # Beregn summen av tallene
30  total = sum(num_list)
31
32  # Beregn gjennomsnittet
33  count = len(num_list)
34  if count > 0:
35      average = total / count
36  else:
37      average = 0
38
39  # Skriv ut resultatet
40  print("Gjennomsnitt:", average)
```

Innkapsling

- Moduler gir mulighet til å kapsle funksjonalitet, slik at du kan skjule kompleksiteten og interne detaljer bak et enkelt grensesnitt.
- Vi tillater altså at en klasse skal kunne tilby funksjoner gjennom et grensesnitt vi selv definerer
- Hvis vi for eksempel ønsker at det skal være mulig å forandre på en instansvariabel etter klassen er opprettet, lager vi en metode for å gjøre dette
- Ønsker vi derimot at det ikke skal være mulig å forandre på en instansvariabel etter klassen er opprettet unngår vi bare å lage en metode for dette

Kort repetisjon av regler for innkapsling

Regler for innkapsling

- Finnes ingen mekanisme for innkapsling i Python, men vi har et sett med regler for å opprettholde konvensjonen om innkapsling
- Alle metoder og instansvariabler som ikke skal kunne brukes utenfor klassedefinisjonen markeres med "_" foran navnet
- Ingen instansvariabler skal aksesseres på utsiden av klassedefinisjonen
- Ingen hjelpemetoder skal aksesseres på utsiden av klassedefinisjonen
- Den eneste måten man skal kunne manipulere eller hente informasjon fra en klasse er gjennom grensesnittet/metodene definert i klassen

Tidligere eksamensoppgaver

Tidligere eksamensoppgaver

- Gjennomgang av prøveeksamen 2022 oppgave 1 og 2

Studentsystem oppgave

Studentsystem oppgave

- Denne uken skal dere starte å kode klassen "Studentsystem"
- I denne klassen skal dere bruke mye av koden dere allerede har skrevet i Fag og Student klassene (gjenbrukbarhet)
- Dere skal også kun forholde dere til det grensesnittet som er definert i klassene Fag og Studenter (innkapsling).
- En av grunnene til at vi lager en Studentsystem klasse er at hvis vi for eksempel ønsker å lage flere Studentsystemer i samme hovedprogram trenger vi ikke skrive samme kode flere ganger for hvert system. All koden vi trenger er allerede skrevet i Studentsystem klassen.
- Hvis vi får et veldig stort hovedprogram er bli blir det også mye mer oversiktlig å lese koden når vi har skjult kompleksiteten til klassene (abstraksjon/innkapsling)
- Ved på legge implementasjonen til Studentsystem i en klasse kan vi også velge hvilke funksjonaliteter vi ønsker å tilby gjennom grensesnittet (innkapsling)