IN1000

UKE 7 - GRUPPE 1

MIAU@IFI.UIO.NO

Plan for dagen

- Klasser og objekter
- Syntaks
- Metoder
- Grensesnitt
- Hvordan bruker man en klasse?
- Hva skjer når man kaller på en metode?
- Innkapsling

Hvorfor bruke klasser?

```
navn = ['Kjersti', 'Vegard', 'Ole']
alder = [50, 31, 5]
høyde = [175, 192, 110]

def feire_bursdag(navn, alder, høyde, indeks):
    alder[indeks] += 1
    print('Hurra', navn[indeks], 'blir',  {alder[indeks]}, 'år!')
    if alder[indeks] < 18:
        høyde[indeks] += 2

feire_bursdag(navn, alder, høyde, 2)
```

Hvorfor bruke klasser?

```
ole = Person("ole", 5, 110)
ole.feire_bursdag()
```

Hvorfor bruke klasser?

```
ole = Person("ole", 5, 110)
ole.feire_bursdag()
```

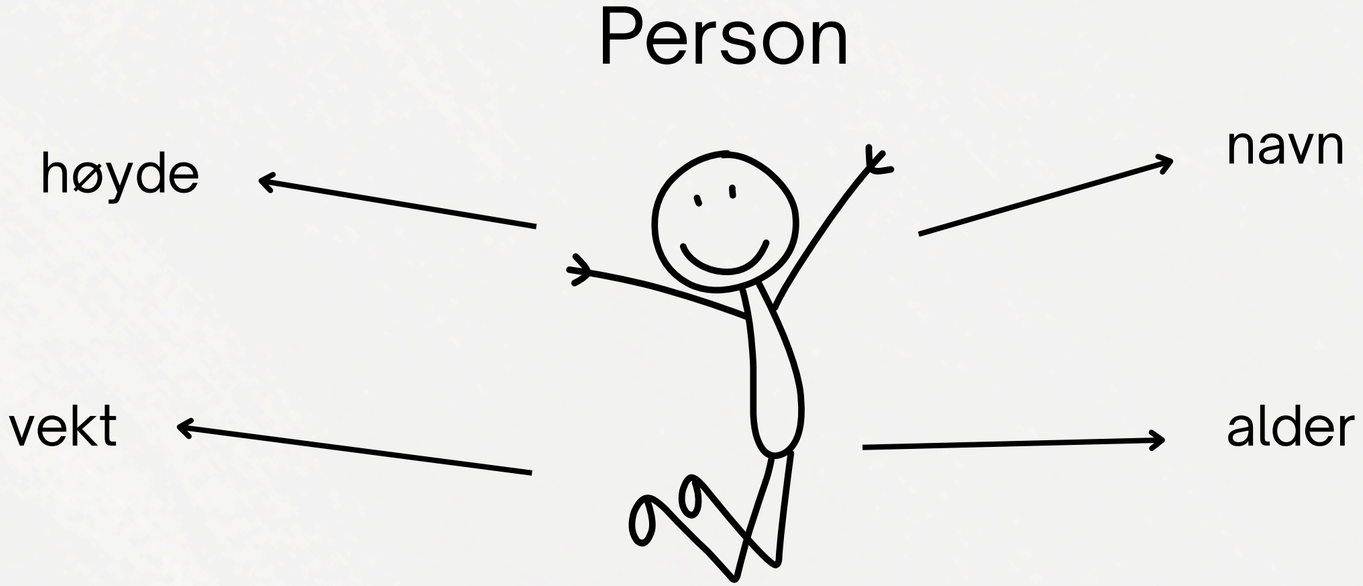
- Kan samle data og funksjon på samme sted
- Slipper å styre med indeks osv.
- Lett å lage så mange personer du vil

Klasser og objekter

- En klasse er en slags mal/oppskrift for noe
  - Forklarer hvordan et objekt skal oppføre seg og hvordan det skal se ut
- Et objekt er en konkret ting laget fra denne oppskriften
- Tenk på en klasse som en pepperkakeform, og et objekt som en pepperkake
  - Alle har samme form, men har ulik pynt

Klasser og objekter

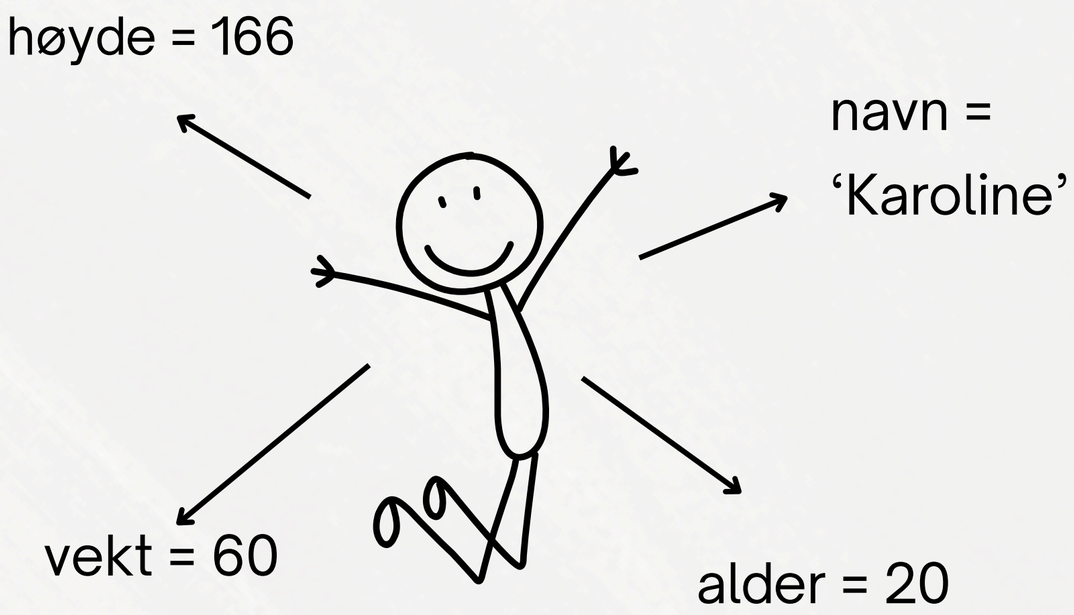
Klassen Person:



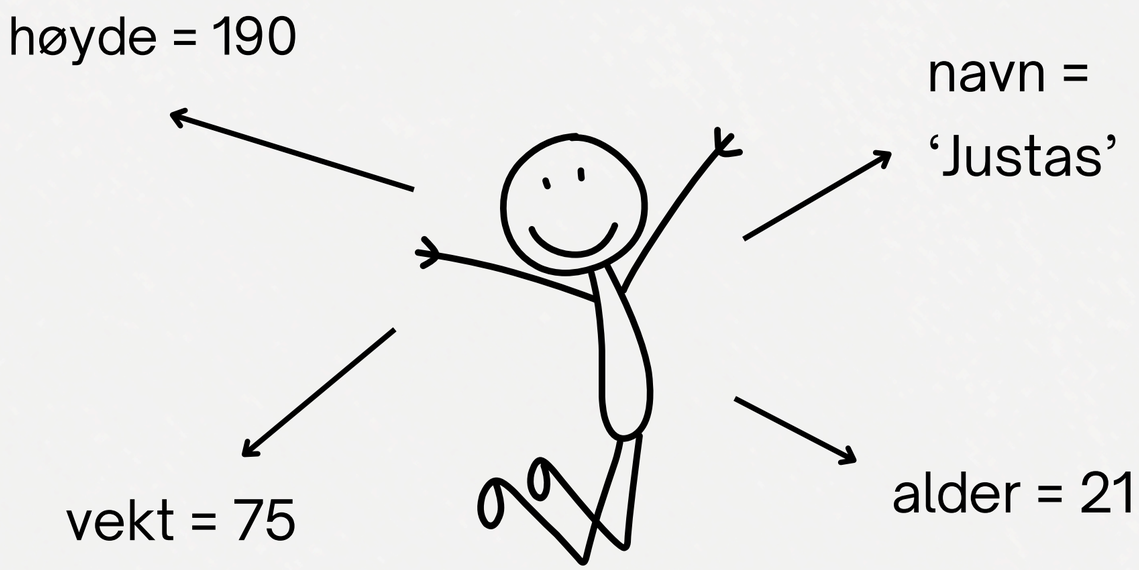
Egenskaper alle objekter av en klasse har (instansvariabler)

Eksempel

Karoline



Justas



Generell syntaks

```
class Klasse:
    def __init__(self, parameter1, parameter2):
        self._egenskap1 = parameter1
        self._egenskap2 = parameter2
```

_ betyr at variabelen bare kan brukes i denne klassen

Oppgave

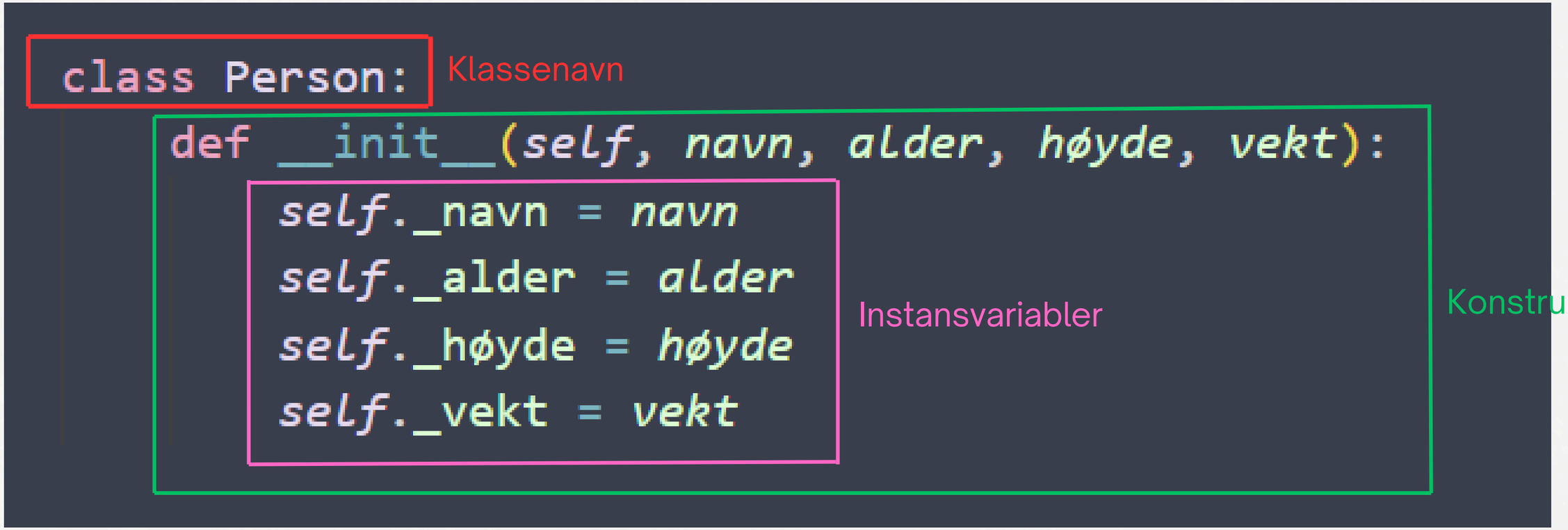
Opprett en fil *person.py*, og lag en klasse ‘Person’ med instansvariablene navn, alder, høyde og vekt

```
class Klasse:
    def __init__(self, parameter1, parameter2):
        self._egenskap1 = parameter1
        self._egenskap2 = parameter2
```

Løsningsforslag

```python
class Person:
    def __init__(self, navn, alder, høyde, vekt):
        self._navn = navn
        self._alder = alder
        self._høyde = høyde
        self._vekt = vekt
```

Løsningsforslag



Parameter eller instansvariabel?

```
def __init__(self, navn, alder, høyde, vekt):
    self._navn = navn
```

navn

- Parameter
- Kan ikke brukes i andre metoder i klassen

self._navn

- Instansvariabel
- Kan brukes i alle metoder innad i klassen

Metoder

- Funksjoner og prosedyrer som er inne i en klasse.

```
class Klasse:
    # konstruktør med instansvariabler

    def metode1(self, parameter1, parameter2):
        # gjør noe

    def metode2(self, paramter1, parameter2):
        # gjør noe
```

Grensesnitt

- Beskriver hva en klasse skal kunne gjøre
  - Hvilke metoder den skal ha
  - Men, ikke hvordan de er implementert
- Et slags skjelett

Grensesnitt

- Beskriver hva en klasse skal kunne gjøre
  - Hvilke metoder den skal ha
  - Men, ikke hvordan de er implementert
- Et slags skjelett

Grensesnitt for Person:

endre_navn(nytt_navn) → endrer navnet til personen

skriv_ut_hilsen() → skriver ut “Hei, jeg heter ... og er ... år gammel”

feire_bursdag() → øker alder med 1, skriver en bursdagshilsen, og øker høyden med 2 hvis person er under 18 år

Oppgave

Du skal nå utvide Person-klassen med følgende metoder:

```
def endre_navn(self, nytt_navn):
    # endrer navn til nytt_navn

def skriv_ut_hilsen(self):
    # skriv ut en hilsen fra personen på formen «Hei, jeg heter
    # <navn> og er <alder> år gammel.»

def feire_bursdag(self):
    # Øk alderen til personen med 1.
    # Skriv ut en bursdagshilsen med den nye alderen («Hurra, <navn>
    # blir <år> år!»).
    # Hvis personen fremdeles er under 18 år, skal høyde økes med 2.
```

Løsningsforslag

```python
class Person:
    def __init__(self, navn, alder, høyde, vekt):
        self._navn = navn
        self._alder = alder
        self._høyde = høyde
        self._vekt = vekt

    def endre_navn(self, nytt_navn):
        self._navn = nytt_navn

    def skriv_ut_hilsen(self):
        print(f'Hei, jeg heter {self._navn} og er {self._alder} år gammel.')

    def feire_bursdag(self):
        self._alder += 1
        print(f'Hurra, {self._navn} blir {self._alder} år!')

        if self._alder < 18:
            self._høyde += 2
```

Hvordan bruker man en klasse?

Du må laste inn klassen fra en fil, til en testfil:

```
# from <filnavn> import <Klasse>
from person import Person
```

Oppretter et objekt av klassen:

```
anne = Person('Anne', 25, 160, 58)
```

Kaller på en metode, på objektet:

```
anne.endre_navn('Dorte')
```

person.py og filen hvor du skal bruke Person må ligge i samme mappe!

Hva skjer når vi kaller en metode?

Selve metodekallet:

```
anne.endre_navn("Dorte")
```

Da vil metoden i praksis se slik ut:

```
# def endre_navn(self, nytt_navn)
#                  ↑         ↑
#                anne      "Dorte"
```

Hva skjer når vi kaller en metode?

- Objektet foran punktumet vil ta plassen til ‘self’ (anne i dette tilfellet)
- Argumentene i parentesen fyller resten av parameterene, som i en vanlig prosedyre/funksjon
- Pga. dette vil man alltid sende en mindre parameter i et metodekall, enn det er skrevet i metode-definisjonen

Innkapsling

- Handler om å skjule data inni et objekt, slik at de bare kan brukes på en kontrollert måte
- _ eller __ foran navnet er en slags avtale mellom programmerere
  - “Ikke bruk denne variabelen utenfor klassen”
  - Du kan teknisk sett fortsatt bruke den, så det er opp til oss å opprettholde den avtalen

Innkapsling

I stedet for å bruke disse variablene, lager vi heller metoder som kan gi oss trygg tilgang og hente de for oss:

```
class Konto:
    def __init__(self, saldo):
        self._saldo = saldo

    def hent_saldo(self):
        return self._saldo

    def sett_inn(self, beløp):
        if beløp > 0:
            self._saldo += beløp
        else:
            print('Beløpet må være positivt!')
```

Finn feilene

Det er 3 feil i denne koden, prøv å finne dem!

```
class Hund:
    def __init__(navn, alder):
        self._navn = navn
        self._alder = alder

    def bjeff(self):
        print(self._navn, " sier voff!")

    def bli_eldre(self):
        _alder = self._alder + 1

fido = Hund('Fido', 3)
fido.bjeff
```

Løsningsforslag

```python
class Hund:
    def __init__(self, navn, alder):
        self._navn = navn
        self._alder = alder

    def bjeff(self):
        print(self._navn, " sier voff!")

    def bli_eldre(self):
        self._alder = self._alder + 1

fido = Hund('Fido', 3)
fido.bjeff()
```



Oppgave

Gitt klassen Person som vi allerede har laget:

1. Opprett 3 objekter av klassen med navn/alder/høyde/vekt
2. Skriv deretter ut en hilsen fra hver person ved hjelp av .skriv_ut_hilsen()
3. Feir bursdagen til en av personene, og skriv ut en ny hilsen fra denne personen
4. Endre navnet til en av personene og skriv ut en ny hilsen fra denne personen

Løsningsforslag

```python
from person import Person

kjersti = Person('Kjersti', 50, 175, 72)
vegard = Person('Vegard', 31, 192, 95)
ole = Person('Ole', 5, 110, 32)

kjersti.skriv_ut_hilsen()
vegard.skriv_ut_hilsen()
ole.skriv_ut_hilsen()

ole.feire_bursdag()
ole.skriv_ut_hilsen()

vegard.endre_navn('Geir')
vegard.skriv_ut_hilsen()
```

Oppgave

Skriv en klasse Dyr. Et dyr har følgende egenskaper: art, kjønn og vekt.

1. Lag en konstruktør
2. Lag en metode som skriver ut informasjonen om dyret
3. Opprett tre objekter av klassen Dyr i en annen fil, for å representere tre forskjellige dyr. Skriv ut informasjonen om alle dyrene