# Gruppetime uke 7
# IN1000

# Plan for timen

- Programmere med objekter
- Objektorientert programmering som konsept
  - Klasser
  - Objekter/instanser av en klasse
- Objektorientert programmering: Teori
  - Hvordan lage en klasse
  - Konstruktør
  - Instanser av en klasse
  - Metoder
  - Innkapsling
  - Hvordan laste inn en klasse til en annen fil

# Programmering med objekter

# Vi allerede brukt objekter i som ligger i Python

- Vi har for eksempel brukt lister
- Når vi lager en liste lager vi et liste objekt
- Som vi har sett tilbyr lister en rekke tjenester som .append() og .pop()
- Nå skal vi lære å lage våre egne klasser og objekter, som kan tilby tjenester vi lager

# Objektorientert programmering som konsept

# Objektorientert programmering

- Handler om å modellere verden rundt deg – i hvert fall en forenklet versjon av den
  - Å modellere handler om å gjenskape egenskaper og/eller oppførselen til et objekt
- En klasse beskriver et sett med objekter med samme oppførsel
- Man kan lage flere objekter av samme klasse
- Vi kan for eksempel ha en klasse som heter bil, som vi kan bruke til å lage flere bil-objekter.

# Objektorientert programmering

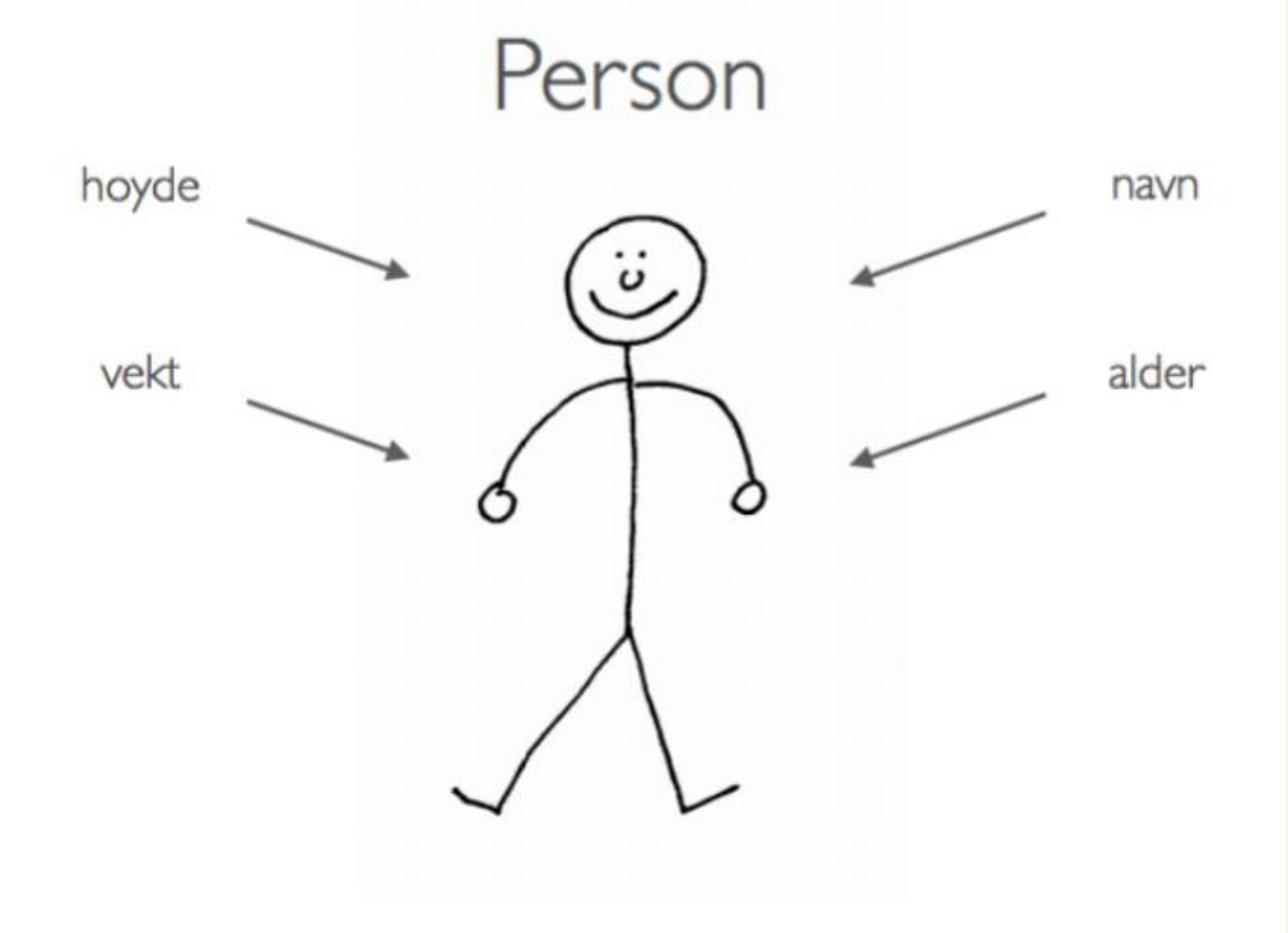
- Handler om å modellere verden rundt deg – i hvert fall en forenklet versjon av den
  - Å modellere handler om å gjenskape egenskaper og/eller oppførselen til et objekt
- En klasse beskriver et sett med objekter med samme oppførsel
- Man kan lage flere objekter av samme klasse
- Vi kan for eksempel ha en klasse som heter bil, som vi kan bruke til å lage flere bil-objekter.
  - Egenskapene til en bil kan f.eks. være registreringsnummer, merke og farge
  - Oppførselen til en bil kan f.eks. være at den kan kjøre og stoppe
  - Videre ønsker vi kanskje å hente ut informasjon fra bilen, som f.eks. hvilket merke bilen er eller om den kjører eller står stille

# Objektorientert programmering

- Handler om å modellere verden rundt deg – i hvert fall en forenklet versjon av den
  - Å modellere handler om å gjenskape egenskaper og/eller oppførselen til et objekt
- En klasse beskriver et sett med objekter med samme oppførsel
- Man kan lage flere objekter av samme klasse
- Vi kan for eksempel ha en klasse som heter bil, som vi kan bruke til å lage flere bil-objekter.
  - Egenskapene til en bil kan f.eks. være registreringsnummer, merke og farge
  - Oppførselen til en bil kan f.eks. være at den kan kjøre og stoppe
  - Videre ønsker vi kanskje å hente ut informasjon fra bilen, som f.eks. hvilket merke bilen er eller om den kjører eller står stille
- Med objektorientert programmering kan vi velge hvilke egenskaper og oppførsel vi vil gi til klassene vil lager, ettersom hva som er nødvendig for at programmet vårt skal funke som det skal

# Eksempel på en klasse – Person

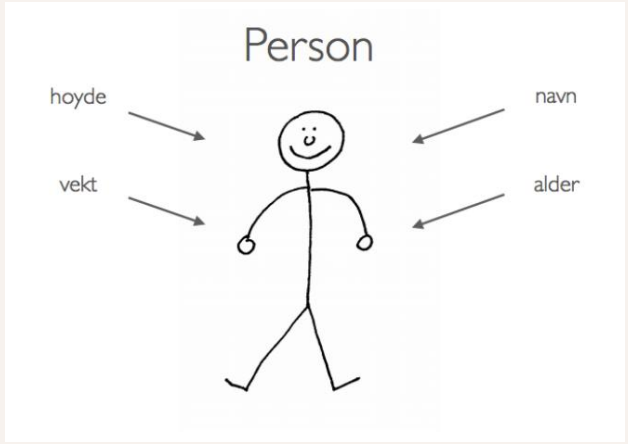
- Når vi lager en klasse må vi velge ut egenskaper vi er interessert i at klassen skal ha
- Når vi for eksempel lager en klasse "Person" er det et utvalg av egenskaper vi ønsker å gi denne klassen



# Forholdet mellom klasser og objekter

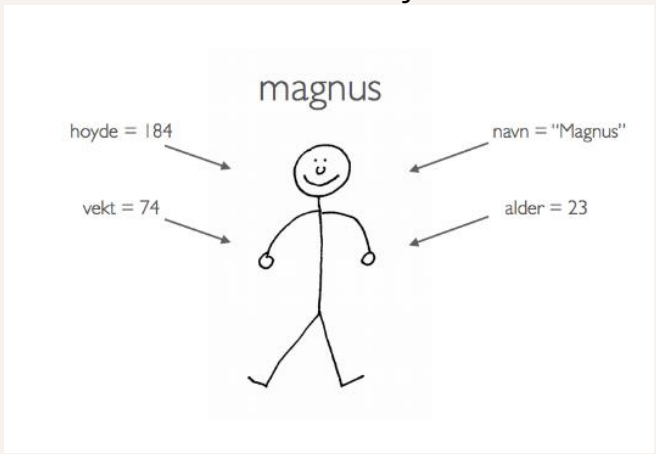
- Alle egenskapene vi ga til person i forrige slide er felles for alle person-objekter
- Person klassen vi nå har laget er kun en **mal**, og ikke en person i seg selv.
- Vi kan bruke klassen Person til å opprette Person-objekter senere
- Når vi lager et Person-objekt sier vi at vi lager en **instans** av klassen person

# Instanser (objekter) av en klassen Person

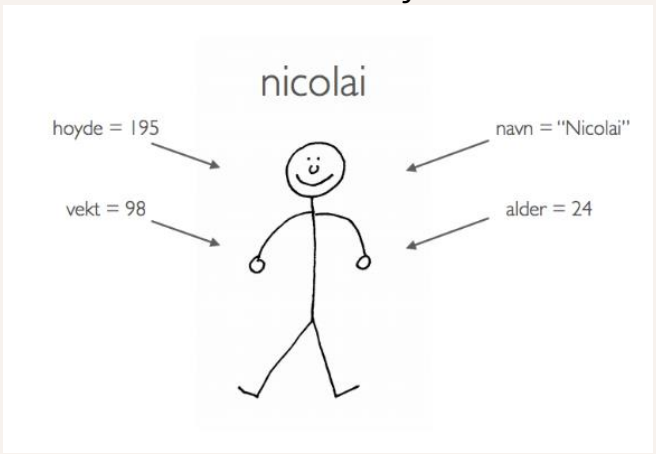



Person Klasse

Person Objekt




Person Objekt



# Forskjellen mellom klasser og objekter

| KLASSE | OBJEKT |
| --- | --- |
| • En klasse brukes som en mal for å kunne lage objekter<br>• Kassen må deklareres før objektet og trenger kun å defineres en gang<br>• I Python bruker man nøkkelordet "class" når man deklarerer en ny klasse<br>• Klasser brukes til på knytte egenskaper og oppførsel til en samling med entiteter/objekter | • Et objekt er en instans av en klasse<br>• Du kan lage så mange objekter av en klasse du vil<br>• Objekter er som variabler av klassen<br>• Hvert objekt har sine egne verdier, f.eks. hver instans av klassen Person har sitt eget navn, alder osv. |

# Objektorientert programmering: Teori

# Hvordan lage en klasse

# Hvordan lage en klasse i Python

- Syntaks:

```
1 class KlasseNavn:
2     # klassens innhold
```

- Her deklarerer vi klassen KlasseNavn. Foreløpig er denne klassen tom og ubrukelig.

- I IN1000 lager vi en klasse per fil

# Konstruktøren til en klasse

# Konstruktøren til en klasse

- Konstruktøren til en klasse bestemmer hva som må med i opprettelsen av et objekt
- Variablene som skal sette verdier i konstruktøren kalles parametere, akkurat som i vanlige funksjoner
- En konstruktør trenger ikke å ha noen parametere
- Syntaks: Konstruktør

```
 8 class Klasse:
 9     def __init__(self, parameter1, parameter2):
10         self.verdi1 = parameter1
11         self.verdi2 = parameter2
```

# Konstruktøren til klasen Person

- I det tidligere eksempelet av klassen Person, skulle en person for eksempel ha et navn, en alder, en høyde og en vekt, da må vi lage en konstruktør som ber om disse egenskapene.
- Vi velger i dette tilfellet å sette alle instansvariablene til et Person objekt til verdiene som blir sendt gjennom parameterne

```
15 class Person:
16     def __init__(self, navn, alder, vekt, hoyde):
17         self.navn = navn
18         self.alder = alder
19         self.vekt = vekt
20         self.hoyde = hoyde
```

# Konstruktøren til klasen Person

- __init__ er en spesialmetode som lar oss lage objekter av klassen. Det er denne metoden som kalles konstruktøren til klassen
- __init__ metoden kalles automatisk når en instans/nytt objekt av en klasse blir opprettet

```
15 class Person:
16     def __init__(self, navn, alder, vekt, hoyde):
17         self.navn = navn
18         self.alder = alder
19         self.vekt = vekt
20         self.hoyde = hoyde
```

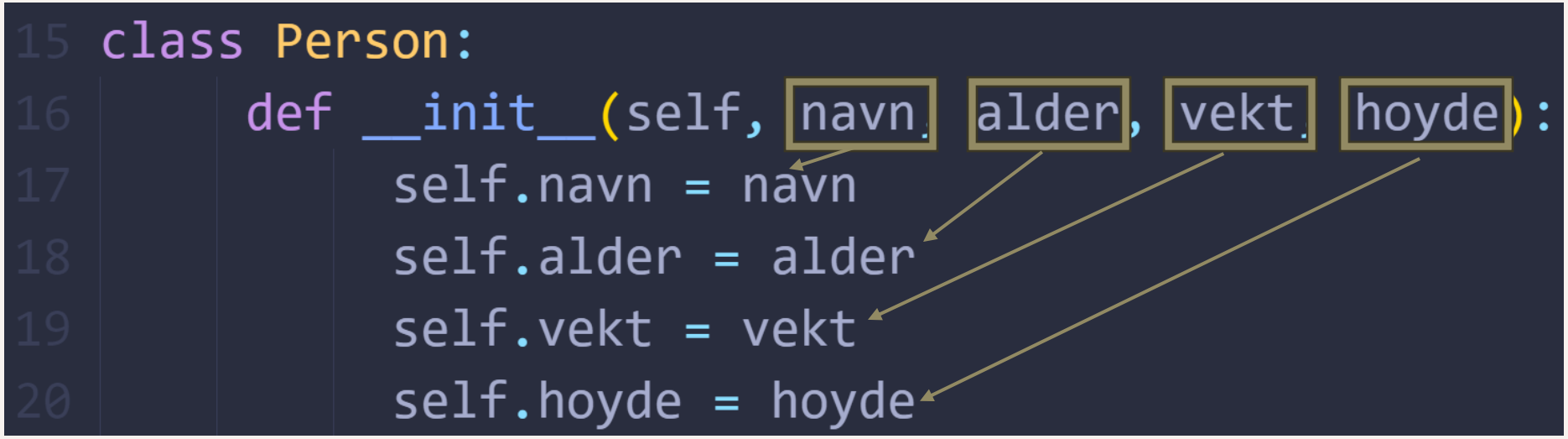
# Konstruktøren til klasen Person

- **Instansvariablene** lagrer informasjonen som ble gitt i opprettelsen av et objekt i variabler som tilhører objektet/en spesifikk instans av klassen
- Alle objekter av klassen har sine egne instansvariabler som er unike for hvert objekt

```
15 class Person:
16     def __init__(self, navn, alder, vekt, hoyde):
17         self.navn = navn
18         self.alder = alder
19         self.vekt = vekt
20         self.hoyde = hoyde
```

# Konstruktøren til klasen Person

- Det er **parameterne** som i dette tilfellet gir verdien til instansvariablene når et objekt av klassen Person blir opprettet
- Vi kunne også hatt instansvariabler som ikke fikk verdiene sine fra et parameter.



# Konstruktøren til klasen Person

- **Self** refererer til det objektet som blir laget
- Self må alltid sendes med som første parameter
- Når vi skriver self.navn, mener vi altså at dette objektet sin navn variabel skal ha verdien til navn-parameteren som ble sendt gjennom konstruktøren

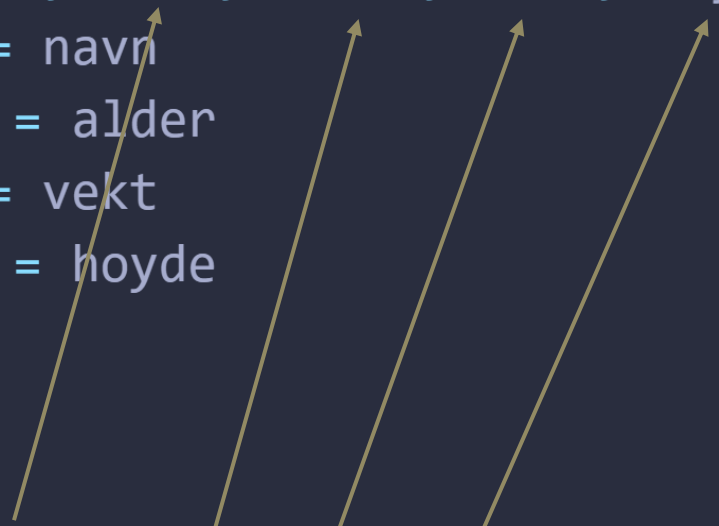
```
15 class Person:
16     def __init__(self, navn, alder, vekt, hoyde):
17         self.navn = navn
18         self.alder = alder
19         self.vekt = vekt
20         self.hoyde = hoyde
```

# Instanser av en klasse

# Instans av klassen Person

- Nå som konstruktøren er på plass kan vi lage en instans (objekt) av klassen person

```
15 class Person:
16     def __init__(self, navn, alder, vekt, hoyde):
17         self.navn = navn
18         self.alder = alder
19         self.vekt = vekt
20         self.hoyde = hoyde
21
22
23
24 person1 = Person("magnus", 23, 74, 184)
```



# Instans av klassen Person

- Vi kan også lage flere person-objekter om vi ønsker

```
15 class Person:
16     def __init__(self, navn, alder, vekt, hoyde):
17         self.navn = navn
18         self.alder = alder
19         self.vekt = vekt
20         self.hoyde = hoyde
21
22
23
24 person1 = Person("magnus", 23, 74, 184)
25 person2 = Person("nikolai", 24, 98, 195)
```

# Oppgave 1: Lag en klasse

- Lag en Klasse som heter **Bil** med instansvariablene **registreringsnummer** og **merke**. Verdien til instansvariablene skal være gitt av parameterne til konstruktøren. Når du har laget Bil-klassen skal du laget et Bil-objekt, altså en instans av bil-klassen.

- Løsningen vises på tavla

# Metoder

# Metoder i klasser

- Så langt har vi bare sett på spesialmetoden __init__(), som er konstruktøren til en klasse
- En klasse kan ha mange metoder
- Vi ønsker at klassen vår skal kunne tilby tjenester gjennom metoder og ikke bare ha egenskaper (instansvariabler). Derfor definerer vi metoder i klassen.
- Et objekt tilbyr kun de tjenestene/metodene som er definert i klassen
- Alle metodene i en klasse har tilgang til instansvariablene
- Alle metoder i en klasse må ta self som første parameter. Dette er igjen for at metoden skal vite hvilket objekt det blir kalt på. Metodene bruker også self for å aksessere instansvariablene til objektene til blir kalt på.
  - NB: Akkurtatt som i __init__ metoden trenger man ikke sende inn et argument for self-parameteren. Dette sendes inn bak kulissene når man kaller en metode til et objekt/oppretter et objekt

# Hvordan lage en metode

- Metoder har samme syntaks som funksjoner og prosedyrer som vi har sett på tidligere.
- Gjennom metodene kan vi for eksempel hente eller forandre instansvariablene i et objekt
- Syntaks:

```
30 class Klasse :
31
32     def metode_navn(self, param1) :
33         #gjør noe
```

# Klasse med metoder: Eksempel

- En Bil klasse med en konstruktør, tre instansvariabler og to metoder

```
29 class Bil:
30     def __init__(self, registreringsNr, merke):
31         self.registreringsNr = registreringsNr
32         self.merke = merke
33         self.km_kjort = 0
34
35     def hent_registreringsNr(self):
36         return self.registreringsNr
37
38     def kjor_km(self, km):
39         self.km_kjort += km
```

# Hvordan bruke metoder fra en klasse

1. Lag et objekt av klassen du har laget
2. Bruk objektet til å aksessere tjenester du har definert i klassen som objektet tilhører

```
41 nyBil = Bil("AA23895", "Skoda")
42
43 nyBil.kjor_km(5)
44 nyBil.hent_registreringsNr()
45 # Kanskje vi vil lagre registreringsnummeret i en variabel
46 regNr = nyBil.hent_registreringsNr()
```

# Innkapsling

# Innkapsling: Konsept

- Når vi lager en klasse ønsker vi å lage et grensesnitt som den som bruker klassen skal kunne anvende
- Det vil si at det er ikke alt innholdet i klassen (metoder og instansvariabler) vi ønsker at skal kunne aksesseres på utsiden av klassen
- Dette er fordi vi ønsker å skille mellom implementasjonen av klassen og grensesnittet til klassen
- En analogi:
  - I en bedrift er det ulike avdelinger, som for eksempel finans, salg, og regnskap. Hver avdeling har sine egne data og ansatte. Hvis noen fra en avdeling trenger data fra en annen avdeling, må de ikke få direkte tilgang til dataene. I stedet må de kontakte en ansatt fra den andre avdelingen og be om dataene de trenger. Dette hjelper med å beskytte dataene og sikre at de bare blir brukt på riktig måte, og dette prinsippet kalles "innkapsling."

# Innkapsling i IN1000

- Python støtter konvensjoner for innkapsling, men har ingen innebygde mekanismer for dette
- Alle metoder som ikke skal inngå i grensesnittet til klassen kalles for hjelpemetoder
- Konvensjonen er da at navnet til disse hjelpemetodene skal starte med "_" – For eksempel def _hjelpemetode():
- En annen konvensjon er at data (instansvariablene) til klassen ikke skal kunne aksesseres utenfor klassen. Vi bruker også her "_" foran instansvariabel-navnene for å indikerer at disse dataene ikke er en del av grensesnittet.
- Eksempel: Bil.py (livekode) - Filen inneholder en Bil klasse med konstruktør og metoder som følger konvensjonene for innkapsling

# Hvordan laste inn en klasse til en annen fil

# Laste inn klasser

- Som tidligere nevnt skal alle klasser være i egen fil i IN1000
- For å laste inn en klasse fra en annen fil må du importere klassen på toppen av filen
- Syntaks:

```
from <filnavn/modul> import <Klasse>
```

- Eksempel for hvis vi ønsker å importere Bil-klassen til en annen fil

```
from Bil import Bil
```

# Oppgave 2: Lag en klasse med metoder

**Oppgave 2**

Vi skal nå utvide Person-klassen vår med følgende metoder:

1. `endre_navn(self, nytt_navn)` → endre navnet til nytt navn.

2. `skriv_ut_hilsen(self)`: → skriv ut en hilsen fra personen, på formen "Hei, jeg heter <navn> og jeg er <alder> år gammel".

3. `feire_bursdag(self)` →
   a. Øk alderen til personen med 1.
   b. Skriv ut en bursdagshilsen med den nye alderen.
   c. Hvis personen fremdeles er under 18 år, skal også høyden økes med 2.

Lag før en Person-klasse med denne konstruktøren- løs så oppgaven

```
15 class Person:
16     def __init__(self, navn, alder, vekt, hoyde):
17         self.navn = navn
18         self.alder = alder
19         self.vekt = vekt
20         self.hoyde = hoyde
```

# Oppgave 3: Lag instanser og bruk metodene

Løsning oppgave 2 og 3: Person.py

Både Person.py og Bil.py legges ut under gruppe18/uke7 på semesteriden

**Oppgave 3**

Gitt klassen Person som vi har laget over. I en egen fil:

1. Opprett 3 instanser (objekter) av klassen med navn/alder/vekt/høyde du velger selv.

2. Skriv deretter ut en hilsen fra hver person deres ved hjelp av skriv_ut_hilsen-metoden.

3. Feire bursdagen til én av personene, og skriv ut en ny hilsen fra denne personen.

4. Endre navnet til én av personene og skriv ut en ny hilsen fra denne personen.

# Jobbe med innlevering/Trix

- Ukens pensum:
  - Trix oppgave 07.01-07.11
    - https://trix.ifi.uio.no/course/8?tags=uke7
  - Innlevering 6

Temaer gjennomgått så langt :

Uke 1:
- Variabler
- Feilmeldinger
- Input fra tastatur
- Beslutninger

Uke 2:
- Data-typer
- Evaluering av utrykk
- Kodeflyt
- Prosedyrer

Uke 3:
- Samlinger og lister
- Mengder
- Ordbøker
- Nøstede samlinger

Uke 4:
- Løkker
- Kombinere løkker og samlinger
- For løkker
- While løkker

Uke 5:
- Lese fra og skrive til filer
- Parametere i prosedyrer
- Retur verdier

Uke 6:
- Skop

Denne uken: Objektorientert programmering