

# Velkommen til gruppetime i IN1000



6. oktober 2021  
Jessie Yue Guan

# Planen for i dag

- Kahoot! :D
- Prosedyreorientert vs. Objektorientert
- Klasser og objekter
- Instansvariabler og instansmetoder
- Self og konstruktøren
- Innkapsling
- Grensesnitt
- Implementasjon

OOP



# Prosedyreorientert vs. Objektorientert

- Hittil har vi bare sett på prosedyreorientert programmering (procedural programming)
  - Bryter ned problemet til en mengde prosedyrer og funksjoner
  - Kan være litt kunstig og litt mer virkelighetsfjernt
  - Passer litt bedre for kalkulasjoner
- Nå skal vi se litt mer på objektorientert programmering (object-oriented programming)
  - Bryter ned problemet til en mengde klasser og objekter
  - Kan være litt mer naturlig og litt mer virkelighetsnært
  - Passer litt bedre for modellering

# Klasser og objekter



Australian  
Cattle Dog



Australian  
Shepherd



Basset  
Hound



Border  
Collie



Boxer



Bulldog



Chow Chow



Dalmatian



Labrador  
Retriever



Pit Bull  
Terrier



Samoyed



Shetland  
Sheepdog



Siberian  
Husky



Vizsla

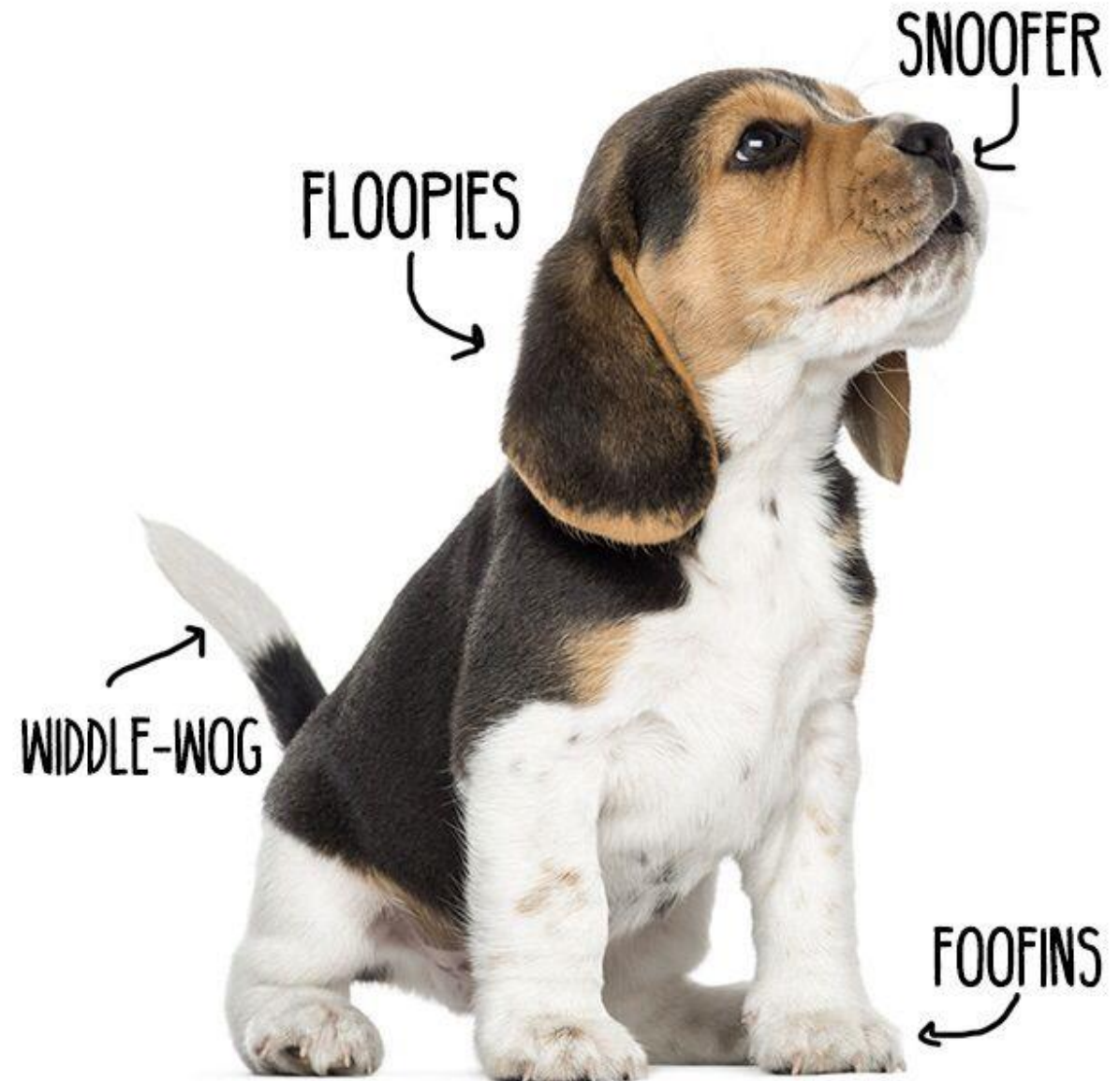


Weimaraner

# Klasser og objekter/instanser

- Klasser er kategorier som inneholder ting
- For eksempel: Dyr, møbel, frukt
- Klasser som er innebygd i Python inkluderer bl.a. primitive variabeltyper
  - String
  - Int
  - Float
  - Boolean
- Objekter er ting som tilhører kategorier
- For eksempel: Katt, stol, eple
- Instanser blir da variablene som dere lager i programmene deres
  - **tekst = "Hei!"**
  - **heltall = 2**
  - **desimaltall = 0.123**
  - **lysbryter = True**

# Instans- variabler



# Attributter/Instansvariabler

- Variabler som tilhører en klasse kalles for attributter/instansvariabler
- Klasser har ulike attributter som beskriver objektene som tilhører klassen
- Attributtene sier noe om hva klassen/objektet er eller hvilke egenskaper den/det har
- For eksempel
  - La oss si at "Hund" er en klasse
  - La oss si at "Hachiko", "Lassie", og "Pluto" er objekter i denne klassen
  - Da kan "navn", "rase", og "farge" være instansvariabler i denne klassen



# Instans- metoder



Stand



Investigate



Shrink



Bark



Attack



Turn away



Yawn



Lick nose



Sniff ground



Walk away



Puppy eyes



Sniff



Scrach



Shake



Blink

# Metoder/Instansmetoder

- Prosedyrer og funksjoner som tilhører en klasse kalles for metoder/instansmetoder
- Klasser har ulike metoder som kan brukes på objektene som tilhører den klassen
- Metoder sier noe om hva klassen/objektet gjør eller hvilke prosesser den/det har
- For eksempel
  - La oss si at "Hund" er en klasse
  - La oss si at "Hachiko", "Lassie", og "Pluto" er objekter i denne klassen
  - Da kan "spise()", "sove()", og "bjeffe()" være instansmetoder i denne klassen

# Syntaks for å opprette klasser

- Legg merke til at klassen opprettes i en egen fil
- Legg merke til at klassenavnet har stor forbokstav
- Legg merke til at filen er liten forbokstav
- Legg merke til at filen heter det samme som klassen
- Instansvariablene er navn, alder, farger, og kjønn
- Instansmetodene er fyllBursdag(), brukTidsmaskin(), og siNoe()

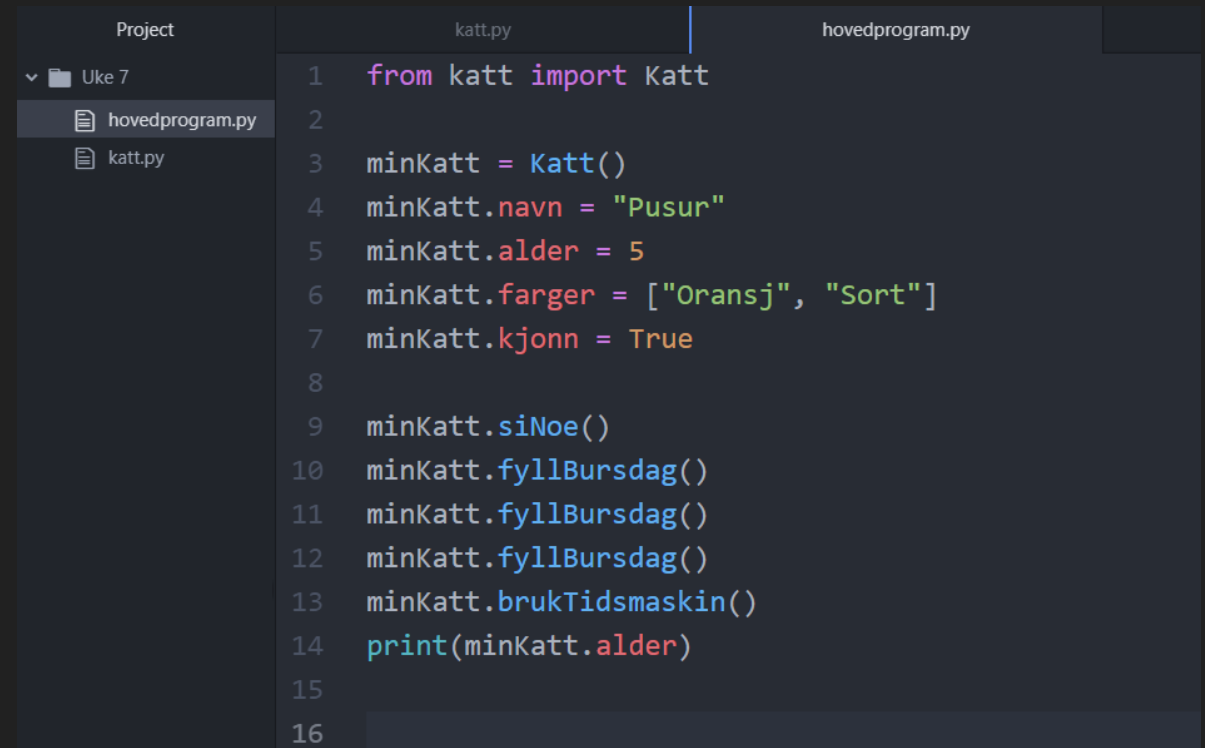


```
Project
└─ Uke 7
   └─ hovedprogram.py
      └─ katt.py

katt.py
1 class Katt():
2     navn = ""
3     alder = 0
4     farger = []
5     kjønn = None
6
7     def fyllBursdag(self):
8         self.alder += 1
9
10    def brukTidsmaskin(self):
11        self.alder -= 1
12
13    def siNoe(self):
14        print("Mjaaau!")
15
```

# Syntaks for å opprette objekter

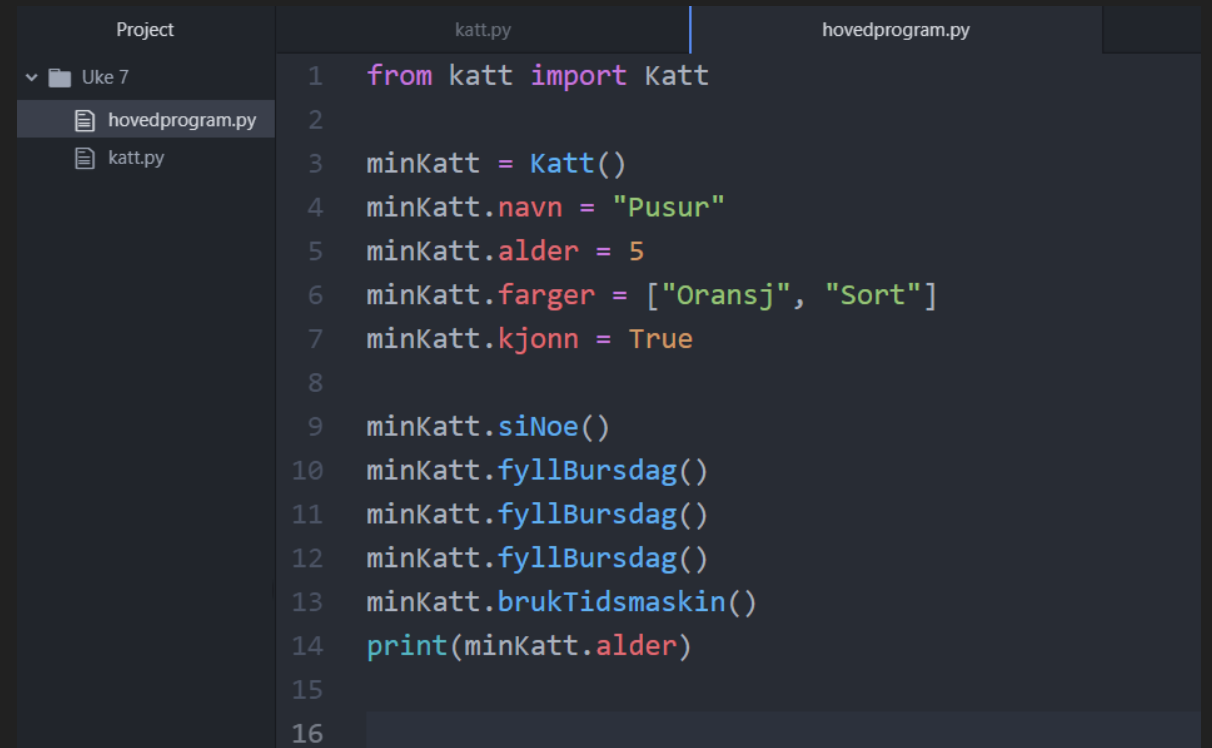
- Legg merke til at objektet opprettes i en egen fil
- Legg merke til at vi importerer klassen Katt fra filen katt
- Legg merke til at filen som oppretter klassen og filen som oppretter objekter ligger i samme mappe
- Her kan vi aksessere og endre instansvariablene
- Her kan vi aksessere og bruke instansmetodene



```
Project
├── Uke 7
│   ├── hovedprogram.py
│   └── katt.py
└── katt.py
    1 from katt import Katt
    2
    3 minKatt = Katt()
    4 minKatt.navn = "Pusur"
    5 minKatt.alder = 5
    6 minKatt.farger = ["Oransj", "Sort"]
    7 minKatt.kjonn = True
    8
    9 minKatt.siNoe()
   10 minKatt.fyllBursdag()
   11 minKatt.fyllBursdag()
   12 minKatt.fyllBursdag()
   13 minKatt.brukTidsmaskin()
   14 print(minKatt.alder)
   15
   16
```

# Syntaks for å opprette objekter

- Legg merke til at vi oppretter et objekt ved å skrive klassenavnet og to parenteser
- Legg merke til at navnet på variabelen som inneholder objektet (minKatt) og navnet til katte-objektet (Pusur) er to forskjellige ting
- Legg merke til at vi bruker instansvariabler og instansmetoder ved å skrive navnet på variabelnavnet til objektet, punktum, og navnet på instansvariabelen eller instansmetoden



```
Project
├── Uke 7
│   ├── hovedprogram.py
│   └── katt.py
└── katt.py
    1 from katt import Katt
    2
    3 minKatt = Katt()
    4 minKatt.navn = "Pusur"
    5 minKatt.alder = 5
    6 minKatt.farger = ["Oransj", "Sort"]
    7 minKatt.kjonn = True
    8
    9 minKatt.siNoe()
   10 minKatt.fyllBursdag()
   11 minKatt.fyllBursdag()
   12 minKatt.fyllBursdag()
   13 minKatt.brukTidsmaskin()
   14 print(minKatt.alder)
   15
   16
```

# Spørsmål?

- Ikke vær redd for å spørre, det finnes ingen dumme spørsmål! 😊

Self



# Hva er self?

- Alt som står innenfor skopet til klassen er felles for alle objekter av den klassen
- Alt som står innenfor skopet til en instansmetode er spesifikt for kun dette objektet
- ALLE instansmetoder MÅ ha minst et parameter self eller så vil ikke programmet fungere
- Nøkkelordet self refererer til dette spesifikke objektet som instansmetoden skal kjøres på



# Hva er self?

- I vårt tilfelle blir for eksempel `self.alder` fra instansmetoden `fillBursdag(self)` erstattet med `minKatt.alder` fra hovedprogrammet
- Dette gjør at Python vet hvilket katte-objekt som fyller bursdag hvis vi har flere katte-objekter
- Kan være spesielt nyttig hvis vi tar imot et annet katte-objekt som parameter for instansmetoden
- Vi kunne for eksempel ha hatt en instansmetode `leggTilSøsken(self, søsken)`
- Hvis vi vil gjøre noe med søsknen kan vi selvfølgelig bare bruke parameteret `søsken`
- Man hvis vi heller vil gjøre noe med katte-objektet vårt kan vi bruke nøkkelordet `self`

# Konstruktøren



# Uten konstruktør

- Det er vanlig å sette instansvariablene til en nullverdi innenfor skopet til klassen først
  - For eksempel, en tom streng, **navn = ""**
  - For eksempel, et tomt tall, **alder = 0**
  - For eksempel, en tom liste, **farger = []**
  - For eksempel, et tomt hva-som-helst, **kjønn = None**
- Det er fordi når vi oppretter klassen så vet vi at alle katter har et navn, men vi vet ikke hva dette navnet er før vi oppretter et objekt, noe vi skal gjøre senere i hovedprogrammet
- Men da må vi huske å overskrive nullverdiene manuelt innenfor hovedprogrammet senere
- Dette problemet har en mer elegant løsning som kalles for konstruktøren. Det er en spesiell metode innenfor en klasse som kjøres automatisk hver gang et objekt av denne klassen opprettes.

# Med konstruktør

- Det er vanlig å ha et parameter for hver instansvariabel innenfor definisjonen til konstruktøren
  - `def __init__(self, n, a, f, k):`
- Det er vanlig å sette instansvariablene til parameterne innenfor skopet til konstruktøren
  - For eksempel, en spesifikk streng, `navn = n`
  - For eksempel, et spesifikt tall, `alder = a`
  - \*\*\*For eksempel, en spesifikk liste, `farger = f`
  - For eksempel, en spesifikk , `kjønn = k`
- Det er fordi det er først når vi oppretter objektet at vi vet hva denne spesifikke katten skal hete, så vi kan sende inn dette spesifikke navnet som argument
  - `minKatt = Katt("Grumpy Cat", 8.5, ["Grå", "Beige"], "hann")`

```
Project
├── Uke 7
│   ├── hovedprogram.py
│   └── katt.py
└── katt.py
1  class Katt():
2      def __init__(self, n, a, f, k):
3          self.navn = n
4          self.alder = a
5          self.farger = f
6          self.kjonn = k
7
8      def fyllBursdag(self):
9          self.alder += 1
10
11     def brukTidsmaskin(self):
12         self.alder -= 1
13
14     def siNoe(self):
15         print("Mjaaau!")
16
```

Project

Uke 7

hovedprogram.py

katt.py

katt.py

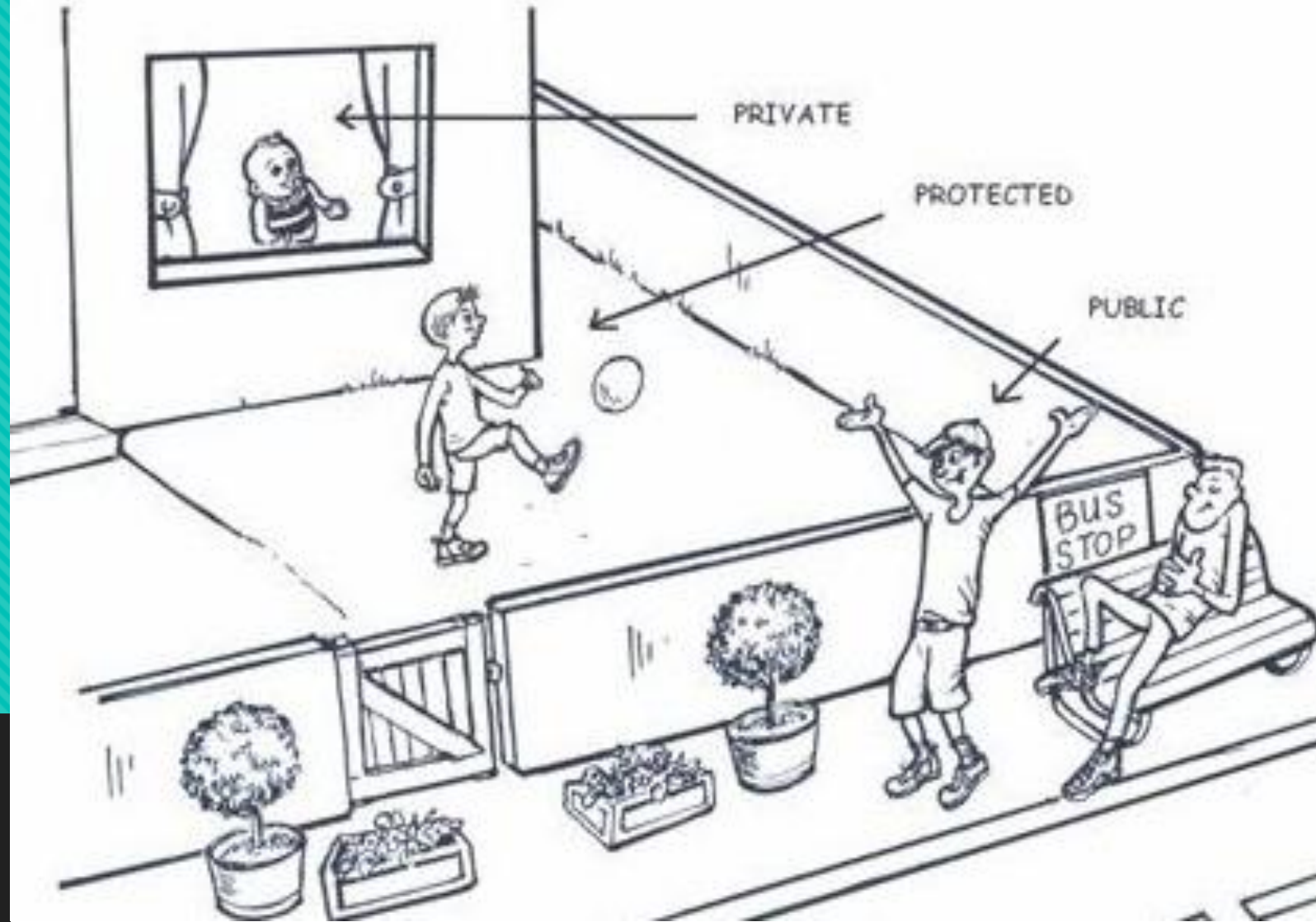
hovedprogram.py

```
1  from katt import Katt
2
3  minKatt = Katt("Pusur", 5, ["Oransj", "Sort"],
4  • True)
5
6  minKatt.siNoe()
7  minKatt.fyllBursdag()
8  minKatt.fyllBursdag()
9  minKatt.fyllBursdag()
10 minKatt.brukTidsmaskin()
11 print(minKatt.alder)
```

# Spørsmål?

- Ikke vær redd for å spørre, det finnes ingen dumme spørsmål! 😊

# Innkapsling





# Public, protected, private

- Public instansvariabler og instansmetoder kan aksesseres overalt
  - **def minMetode(self):**
    - **self.mittAtributt = "Ola Nordmann"**
- Protected instansvariabler og instansmetoder starter med en understrek og skal bare aksesseres innenfor klassen og subclassene til klassen
  - **def \_minMetode(self):**
    - **self.\_mittAtributt = "olanor@uio.no"**
- Private instansvariabler og instansmetoder starter med to understreker og kan kun aksesseres innenfor klassen
  - **def \_\_minMetode(self):**
    - **self.\_\_mittAtributt = "passord123"**

# Getters og Setters

- For hver protected/private instansvariabel er det vanlig å ha to metoder
  - En getter som lar folk aksessere instansvariabelen
    - Den tar imot nøyaktig null parametere
    - Den returnerer bare attributtet
    - Den heter alltid 'get' eller 'hent' også variabelnavnet
  - En setter som lar folk endre instansvariabelen
    - Den tar imot nøyaktig ett parameter
    - Den setter attributtverdien til argumentet
    - Den heter alltid 'set' eller 'sett' også variabelnavnet

```
Project
├── Uke 7
│   ├── hovedprogram.py
│   └── katt.py
└── katt.py
    1 class Katt():
    2     def __init__(self, n, a, f, k):
    3         self.__navn = n
    4         self.__alder = a
    5         self.__farger = f
    6         self.__kjonn = k
    7
    8     def getAlder(self):
    9         return self.__alder
   10
   11     def setAlder(self, nyAlder):
   12         self.__alder = nyAlder
   13
   14     def siNoe(self):
   15         print("Mjaaau!")
   16
```

Project	katt.py	hovedprogram.py
▼ Uke 7	1 from katt import Katt	
hovedprogram.py	2	
katt.py	3 minKatt = Katt("Pusur", 5, ["Oransj", "Sort"], • True) 4 5 minKatt.siNoe() 6 minKatt.setAlder(10) 7 print(minKatt.getAlder()) 8	

# Innkapsling

- Innkapsling innebærer å beskytte sensitive data & prosesser ved å pakke det inn i en boks slik at kun autoriserte brukere/kodere har adgang til dataene & prosessene inni boksen
- På et universitet har vi personalrom, vaktmesterrom, rengjøringsrom, tekniske rom, osv. som uvedkommende uten adgangskort ikke har tilgang til
- Vanlige mennesker trenger ikke å vite hva som er i rommene eller hva som skjer inni dem, de trenger bare å vite at de fungerer som de skal
- Innkapsling kan være mye forskjellig, men det refererer vanligvis til bruk av private/protected instansvariabler og getters/setters innenfor en klasse

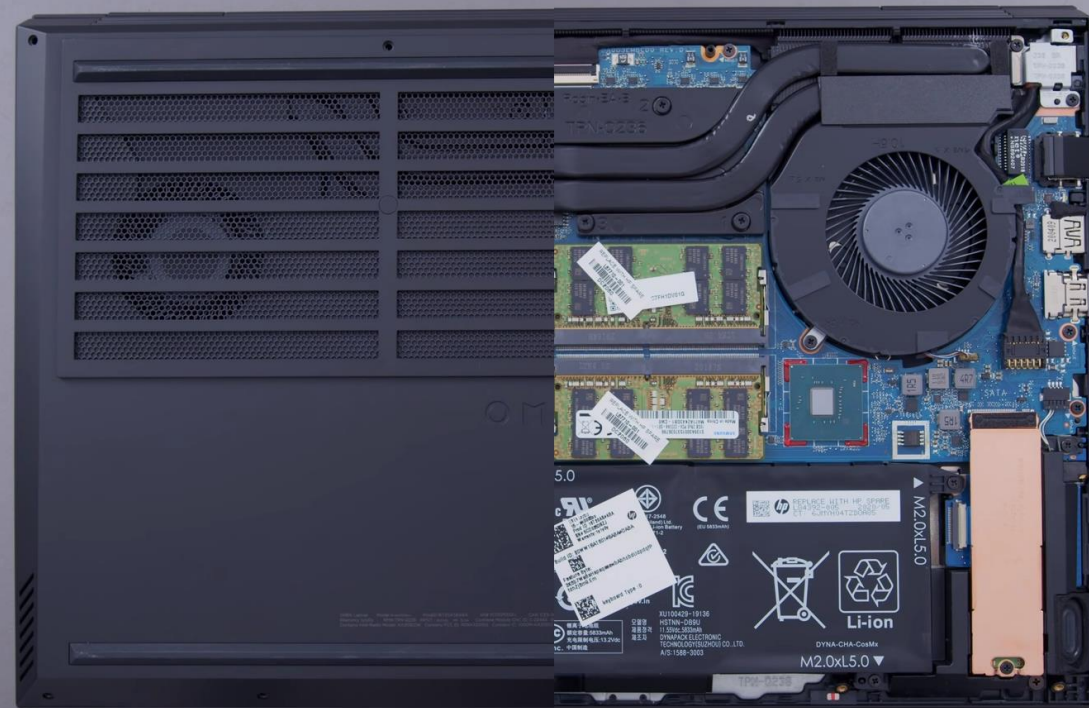
```
Project
├── Uke 7
│   ├── hovedprogram.py
│   └── katt.py
└── katt.py

1 class Katt():
2     def __init__(self, n, a, f, k):
3         self.__navn = n
4         self.__alder = a
5         self.__farger = f
6         self.__kjonn = k
7
8     def getAlder(self):
9         print("Kontrollspørsmål!")
10        svar = input("Hva heter katten?")
11        if svar == self.__navn:
12            return self.__alder
13
14        def setAlder(self, nyAlder):
15            print("Kontrollspørsmål!")
16            svar = input("Hva heter katten?")
17            if svar == self.__navn:
18                self.__alder = nyAlder
19
20        def siNoe(self):
21            print("Mjaaau!")
22
```

# Spørsmål?

- Ikke vær redd for å spørre, det finnes ingen dumme spørsmål! 😊

# Grensesnitt/ Implemen- tasjon





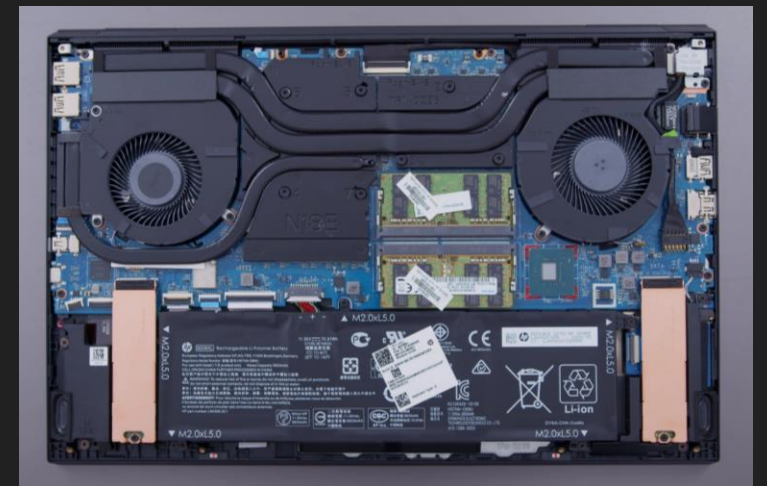
# Grensesnitt og implementasjon

- Alle klasser har et grensesnitt (interface) og en implementasjon (implementation)
- Grensesnittet består av navnet, parameterne, og returtypen til alle metodene i en klasse
- Implementasjonen består av kodeblokkene som er innenfor skopet til alle metodene i en klasse



← Grensesnitt

Implementasjon →



```
Project
├── Uke 7
│   ├── hovedprogram.py
│   └── katt.py
└── katt.py

1 class Katt():
2     def __init__(self, n, a, f, k):
3         self.__navn = n
4         self.__alder = a
5         self.__farger = f
6         self.__kjonn = k
7
8     def getAlder(self):
9         print("Kontrollspørsmål!")
10        svar = input("Hva heter katten?")
11        if svar == self.__navn:
12            return self.__alder
13
14        def setAlder(self, nyAlder):
15            print("Kontrollspørsmål!")
16            svar = input("Hva heter katten?")
17            if svar == self.__navn:
18                self.__alder = nyAlder
19
20        def siNoe(self):
21            print("Mjaaau!")
22
```

```
Project
├── Uke 7
│   ├── hovedprogram.py
│   └── katt.py
└── katt.py
1 class Katt():
2     def __init__(self, n, a, f, k):
3         self.__navn = n
4         self.__alder = a
5         self.__farger = f
6         self.__kjonn = k
7
8     def getAlder(self):
9         print("Kontrollspørsmål!")
10        svar = input("Hva heter katten?")
11        if svar == self.__navn:
12            return self.__alder
13
14        def setAlder(self, nyAlder):
15            print("Kontrollspørsmål!")
16            svar = input("Hva heter katten?")
17            if svar == self.__navn:
18                self.__alder = nyAlder
19
20        def siNoe(self):
21            print("Mjaaau!")
22
```

# Spørsmål?

- Ikke vær redd for å spørre, det finnes ingen dumme spørsmål! 😊

# Innkapsling versus grensesnitt/implementasjon

- Innkapsling er det å skjule instansvariabler i klassen din fra andre ved å ha egne metoder som bestemmer hvem, når, hvor, hvordan, osv. folk kan aksessere eller endre de.
- Innkapsling er med andre ord de kodelinjene hvor dere setter instansvariablene eller instansmetodene deres til å være private eller protected og de kodelinjene hvor dere definerer metodene for å aksessere (getters) og eventuelt endre (setters) de.

# Innkapsling versus grensesnitt/implementasjon

- Grensesnittet til en klasse er hvilke metoder den inneholder, hvilke parameter de har, og hva de returnerer.
- Grensesnittet er med andre ord de kodelinjene som starter på def og de kodelinjene som starter på return i en klasse.
- Implementasjonen til en klasse er hvordan metodene blir implementert
- Implementasjon er med andre ord de kodeblokkene som ligger indentert under metodene

# Innkapsling versus grensesnitt/implementasjon

- Alle klasser har et grensesnitt og en implementasjon så lenge de inneholder minst en instansmetode
- Men ikke alle klasser har innkapsling med mindre de har private eller protected instansvariabler og egne metoder for å aksessere (getters) eller endre (setters) de.
- Til høyre har vi et eksempel på en klasse med grensesnitt og implementasjon men uten innkapsling

```
1 class Hund:
2
3     def __init__(vekt):
4         self.vekt = vekt
5         self.metthet = 10
6
7     def bjeff():
8         print("Voff voff!")
9
10    def knurr():
11        print("Grrrr!")
```

```
Project
├── Uke 7
│   ├── hovedprogram.py
│   └── katt.py
└── katt.py

1 class Katt():
2     def __init__(self, n, a, f, k):
3         self.__navn = n
4         self.__alder = a
5         self.__farger = f
6         self.__kjonn = k
7
8     def getAlder(self):
9         print("Kontrollspørsmål!")
10        svar = input("Hva heter katten?")
11        if svar == self.__navn:
12            return self.__alder
13
14        def setAlder(self, nyAlder):
15            print("Kontrollspørsmål!")
16            svar = input("Hva heter katten?")
17            if svar == self.__navn:
18                self.__alder = nyAlder
19
20        def siNoe(self):
21            print("Mjaaau!")
22
```



```
Project
├── Uke 7
│   ├── hovedprogram.py
│   └── katt.py
└── katt.py

1 class Katt():
2     def __init__(self, n, a, f, k):
3         self.__navn = n
4         self.__alder = a
5         self.__farger = f
6         self.__kjonn = k
7
8     def getAlder(self):
9         print("Kontrollspørsmål!")
10        svar = input("Hva heter katten?")
11        if svar == self.__navn:
12            return self.__alder
13
14        def setAlder(self, nyAlder):
15            print("Kontrollspørsmål!")
16            svar = input("Hva heter katten?")
17            if svar == self.__navn:
18                self.__alder = nyAlder
19
20        def siNoe(self):
21            print("Mjaaau!")
22
```

```
Project
├── Uke 7
│   ├── hovedprogram.py
│   └── katt.py
└── katt.py
1 class Katt():
2     def __init__(self, n, a, f, k):
3         self.__navn = n
4         self.__alder = a
5         self.__farger = f
6         self.__kjonn = k
7
8     def getAlder(self):
9         print("Kontrollspørsmål!")
10        svar = input("Hva heter katten?")
11        if svar == self.__navn:
12            return self.__alder
13
14        def setAlder(self, nyAlder):
15            print("Kontrollspørsmål!")
16            svar = input("Hva heter katten?")
17            if svar == self.__navn:
18                self.__alder = nyAlder
19
20        def siNoe(self):
21            print("Mjaaau!")
22
```

# Oppdelingen av en klasse

- Linje 1 er klassenavnet til klassen
- Linje 2-4 er instansvariablene til klassen
- Linje 6-9 er konstruktøren til klassen
- Linje 11-18 er instansmetodene til klassen
- Linje 11, 14, 17 blir grensesnittet til klassen
- Linje 12, 15, 18 blir implementasjonen til klassen

```
1 class Katt:
2     navn = ""
3     rase = ""
4     vekt = 0
5
6     def __init__(self):
7         navn = input("Hva heter katten? ")
8         rase = input("Hvilken rase er den? ")
9         vekt = input("Hvor mye veier den? ")
10
11     def spis():
12         vekt += 30
13
14     def sov():
15         vekt -= 30
16
17     def lyd():
18         return "Mjaaaau!"
```

# Spørsmål?

- Ikke vær redd for å spørre, det finnes ingen dumme spørsmål! 😊

# På tide å sjekke om dere har fulgt med...

- Kahoot! :D