Velkommen til gruppetime i IN1000 ☺

02. oktober 2020

Jessie Yue Guan

Planen for i dag

- Prosedyreorientert versus objektorientert programmering
- Klasser og objekter/instanser
- Attributter/instansvariabler og metoder/instansmetoder
- Konstruktøren og innkapsling
- Grensesnitt versus implementasjon

Prosedyreorientert programmering vs. Objektorientert programmering

- Hittil har vi bare sett på prosedyreorientert programmering (procedural programming)
  - Bryter ned problemet til en mengde prosedyrer og funksjoner
  - Pleier å 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 menge klasser og objekter
  - Pleier å være litt mer naturlig og litt mer virkelighetsnært
  - Passer litt bedre for modellering

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**

Attributter/Instansvariabler

- Variabler og kolleksjoner 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

Metoder/Instansmetoder

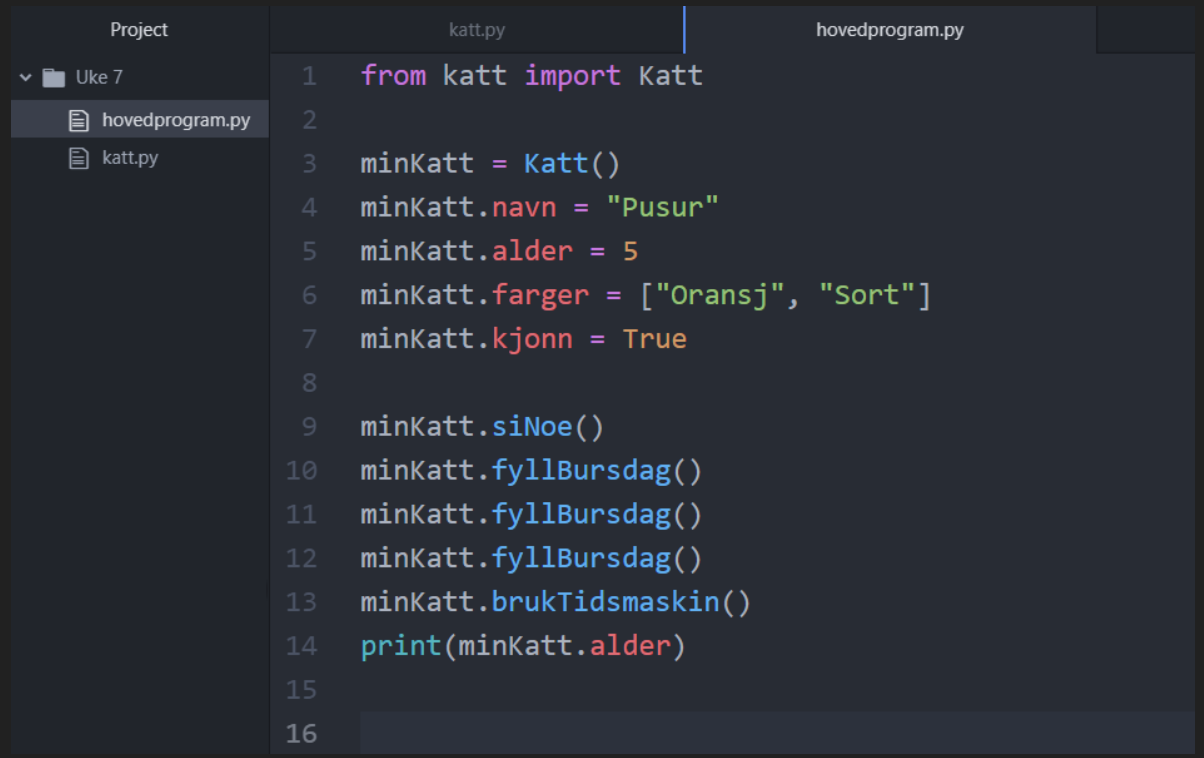
- Prosedyrer og funksjoner som tilhører en klasse kalles for metoder/instansmetoder
- Klasser har ulike metoder som kan brukes på objektene som tilhører den klassen
- Metoder sier noe om hva klassen/objektet gjør eller hvilke prosesser den/det har
- For eksempel
  - La oss si at "Hund" er en klasse
  - 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 at liten forbokstav
- Legg merke til at filen heter det samme som klassen
- Instansvariablene er navn, alder, farger, og kjonn
- Instansmetodene er fyllBursdag(), brukTidsmaskin(), og siNoe()

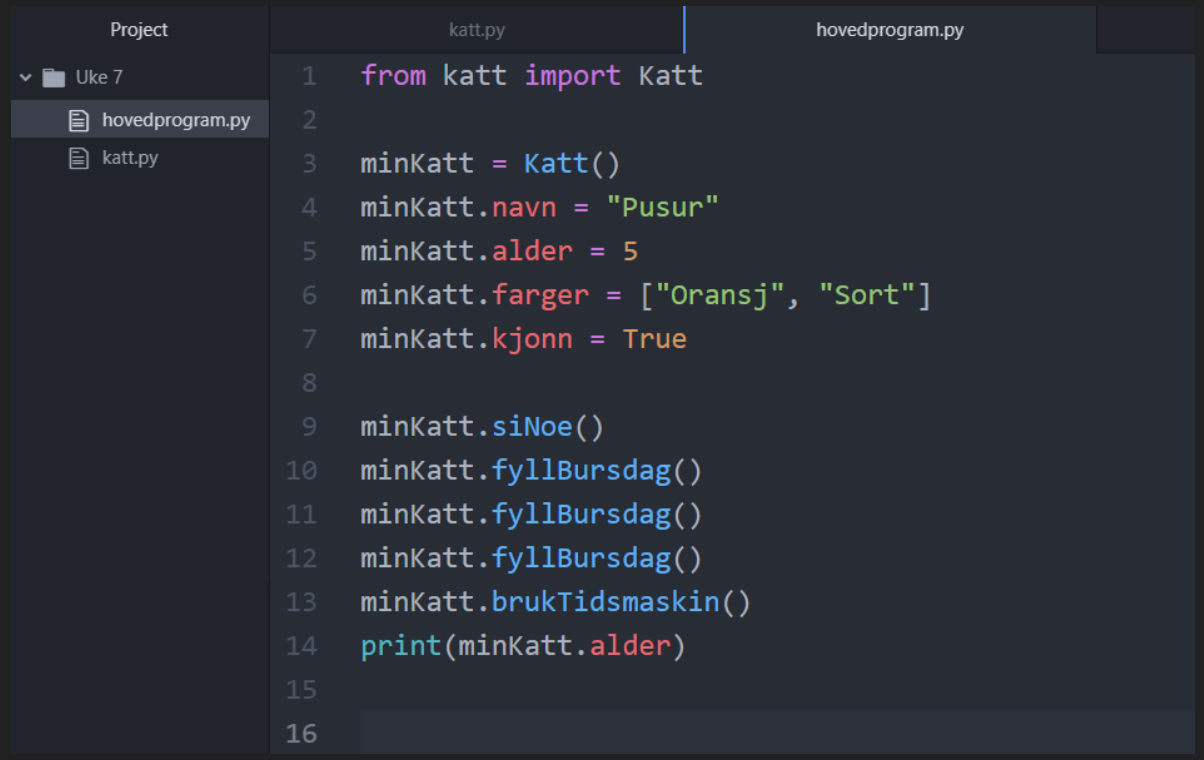
Syntaks for å opprette objekter

- 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



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 variabelen som inneholder objektet, punktum, og navnet på instansvariabelen eller instansmetoden



Hva er self?

- ALLE metodene i en klasse MÅ ha minst et parameter som heter self i Python
- Dette er kjempeviktig, hvis du glemmer det vil ikke programmet ditt fungere!
- Nøkkelordet self referer til dette spesifikke objektet og må brukes for alt som har noe med dette spesifikke objektet å gjøre
- Så for eksempel alder har noe med dette spesifikke objektet å gjøre, så vi må skrive self.alder hvis vi skal aksessere eller endre det
- Vi bruker KUN self i klasser, i hovedprogrammer bruker vi variabelnavn for å referere til objekter/instanser
- Det som skjer når vi kjører metoden fyllBursdag() i hovedprogrammet er at self.alder blir erstattet med minHund.alder i klasseprogrammet

Konstruktøren

- 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 som vi skal gjøre senere i hovedprogrammet
- Dette problemet løses ved hjelp av noe som heter konstruktøren som er en spesiell metode innenfor en klasse som kjører hver gang et objekt av denne klassen opprettes

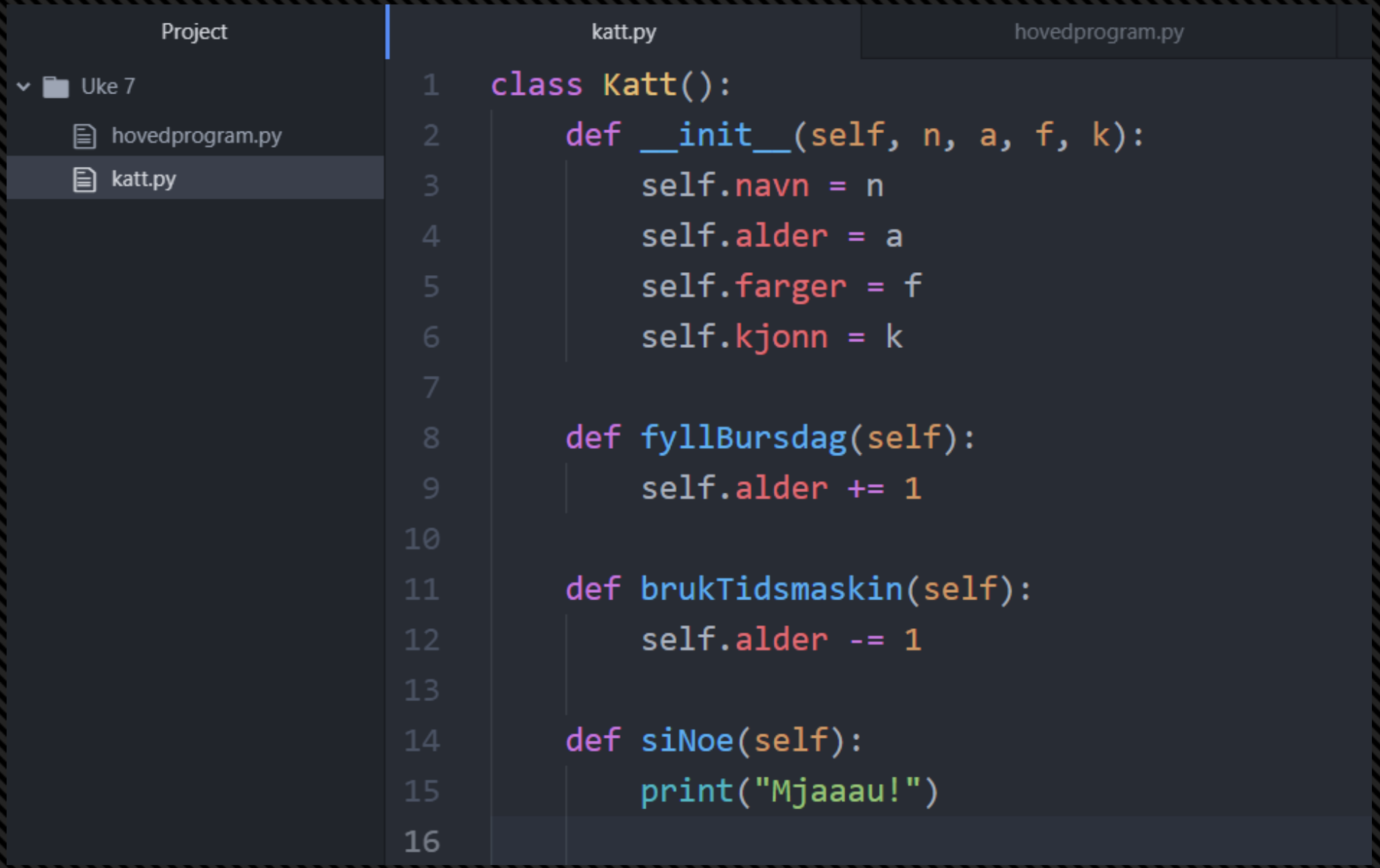
Konstruktøren

- 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 senere
  - 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 hunden skal hete, så vi kan sende inn dette spesifikke navnet som argument
  - **minKatt = Katt("Grumpy", 8.5, ["Grå", "Beige"], "hann")**

- Når vi oppretter et objekt så vet vi at konstruktøren kommer til å kjøre, så vi vet at instansvariablene blir satt til argumentene eller de spesifikke verdiene og nullverdiene overskrives


Project
Uke 7
hovedprogram.py
katt.py
katt.py
hovedprogram.py
class Katt():
def __init__(self, n, a, f, k):
self.navn = n
self.alder = a
self.farger = f
self.kjonn = k
def fyllBursdag(self):
self.alder += 1
def brukTidsmaskin(self):
self.alder -= 1
def siNoe(self):
print("Mjaaau!")

Project

Uke 7

hovedprogram.py

katt.py

katt.py

hovedprogram.py

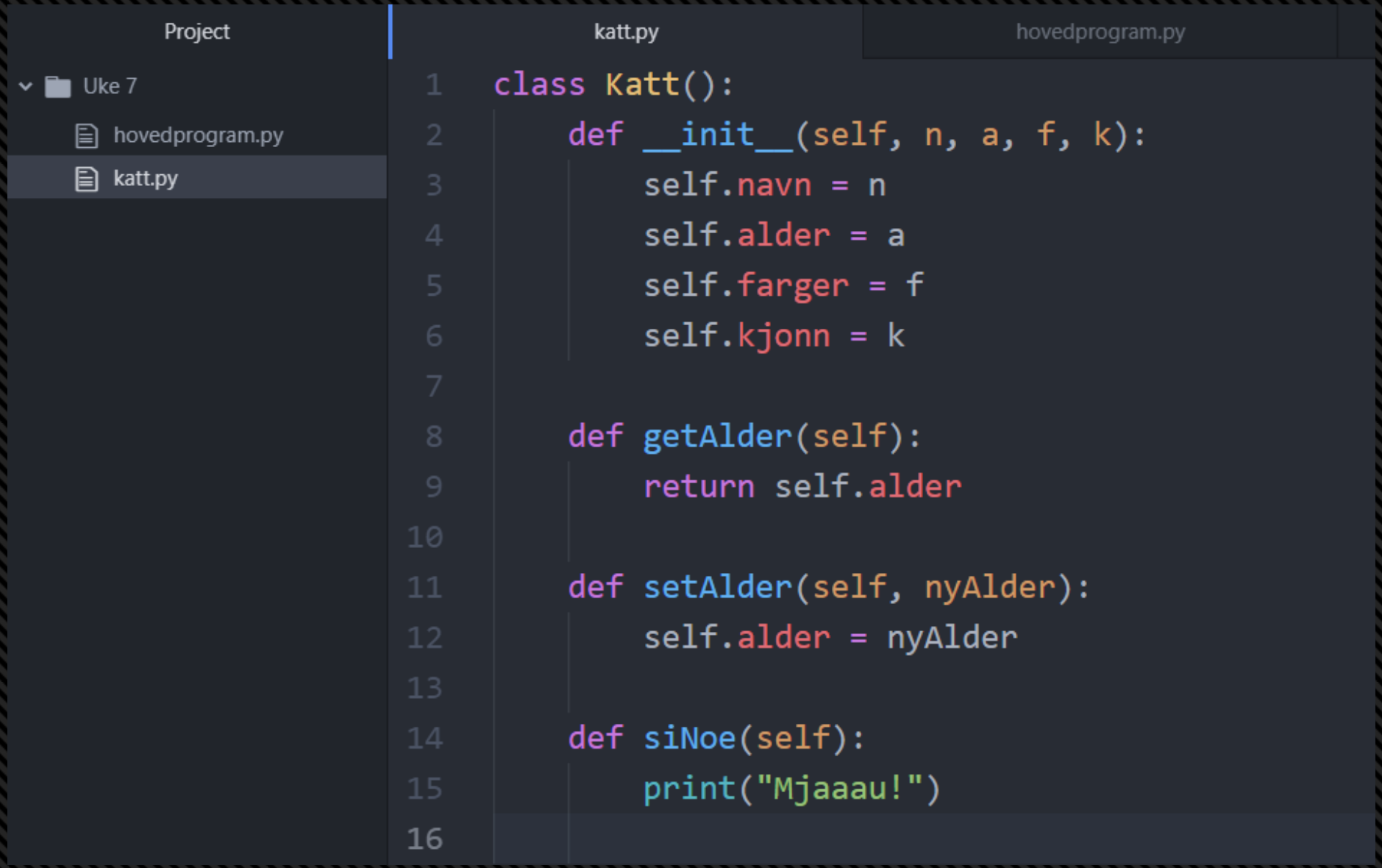
```python
from katt import Katt

minKatt = Katt("Pusur", 5, ["Oransj", "Sort"],
True)

minKatt.siNoe()
minKatt.fyllBursdag()
minKatt.fyllBursdag()
minKatt.fyllBursdag()
minKatt.brukTidsmaskin()
print(minKatt.alder)

```

Public, protected, private

- Public instansvariabler og instansmetoder kan aksesseres innenfor samme mappe
  - **def minProsedyre(self):**
    - **self.minVariabel = «Ola Nordmann»**
- Protected instansvariabler og instansmetoder starter med en understrek og kan bare aksesseres innenfor klassen og alle dens subklasser
  - **def _minProsedyre(self):**
    - **self._minVariabel = «Kari Nordmann»**
- Private instansvariabler og instansmetoder starter med to understreker og kan bare aksesseres innenfor klassen
  - **def __minProsedyre(self):**
    - **self.__minVariabel = «Espen Askeladd»**

Getters og Setters

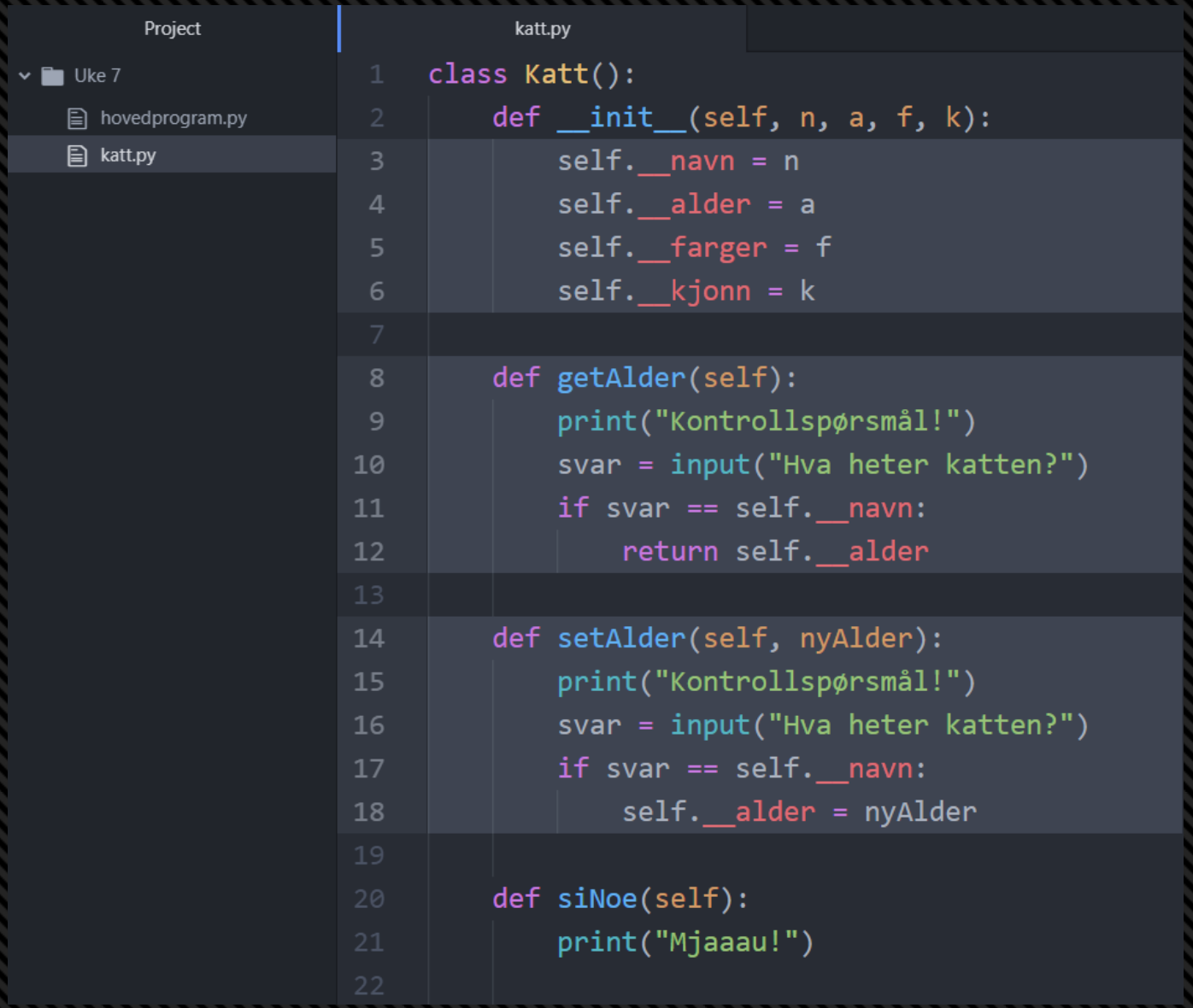
- For hver private/protected instansvariabel er det vanlig å ha to metoder
  - En getter som lar folk aksessere instansvariabelen
    - Den tar imot nøyaktig null parametere
    - Den returnerer bare attributtet
    - Den heter alltid ‘get’ også variabelnavnet
  - En setter som lar folk endre instansvariabelen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


Project
Uke 7
hovedprogram.py
katt.py
katt.py
hovedprogram.py
class Katt():
def __init__(self, n, a, f, k):
self.navn = n
self.alder = a
self.farger = f
self.kjonn = k
def getAlder(self):
return self.alder
def setAlder(self, nyAlder):
self.alder = nyAlder
def siNoe(self):
print("Mjaaau!")


Project
Uke 7
hovedprogram.py
katt.py
katt.py
hovedprogram.py
1 from katt import Katt
2
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 konstruktører/getters/setters innenfor en klasse

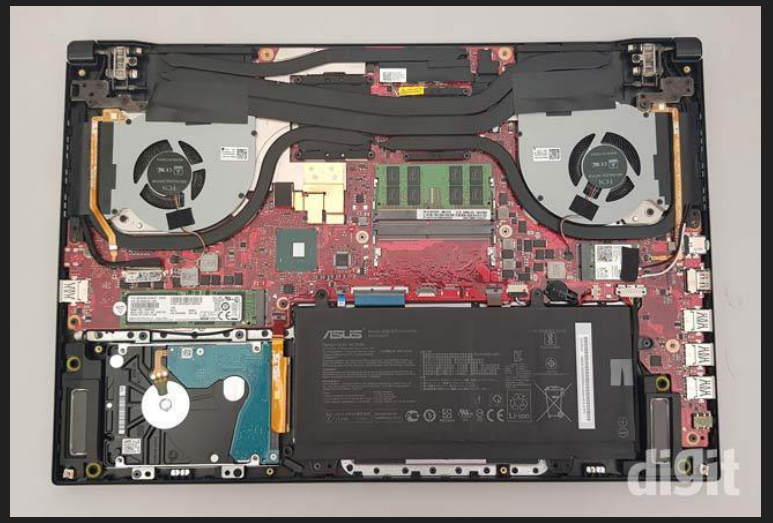

Project
Uke 7
hovedprogram.py
katt.py
katt.py
class Katt():
def __init__(self, n, a, f, k):
self.__navn = n
self.__alder = a
self.__farger = f
self.__kjonn = k
def getAlder(self):
print("Kontrollspørsmål!")
svar = input("Hva heter katten?")
if svar == self.__navn:
return self.__alder
def setAlder(self, nyAlder):
print("Kontrollspørsmål!")
svar = input("Hva heter katten?")
if svar == self.__navn:
self.__alder = nyAlder
def siNoe(self):
print("Mjaaau!")

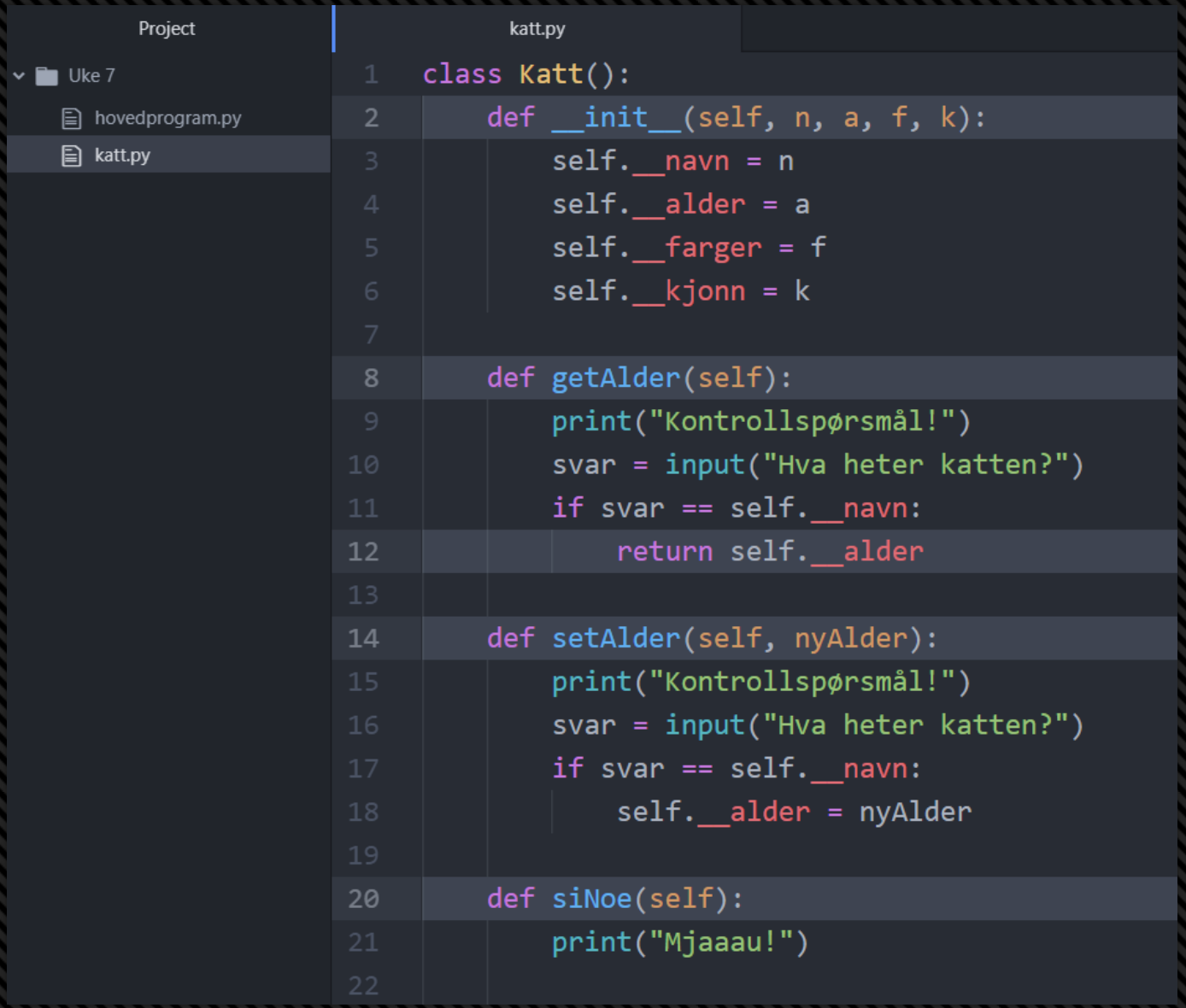
Grensesnitt og implementasjon

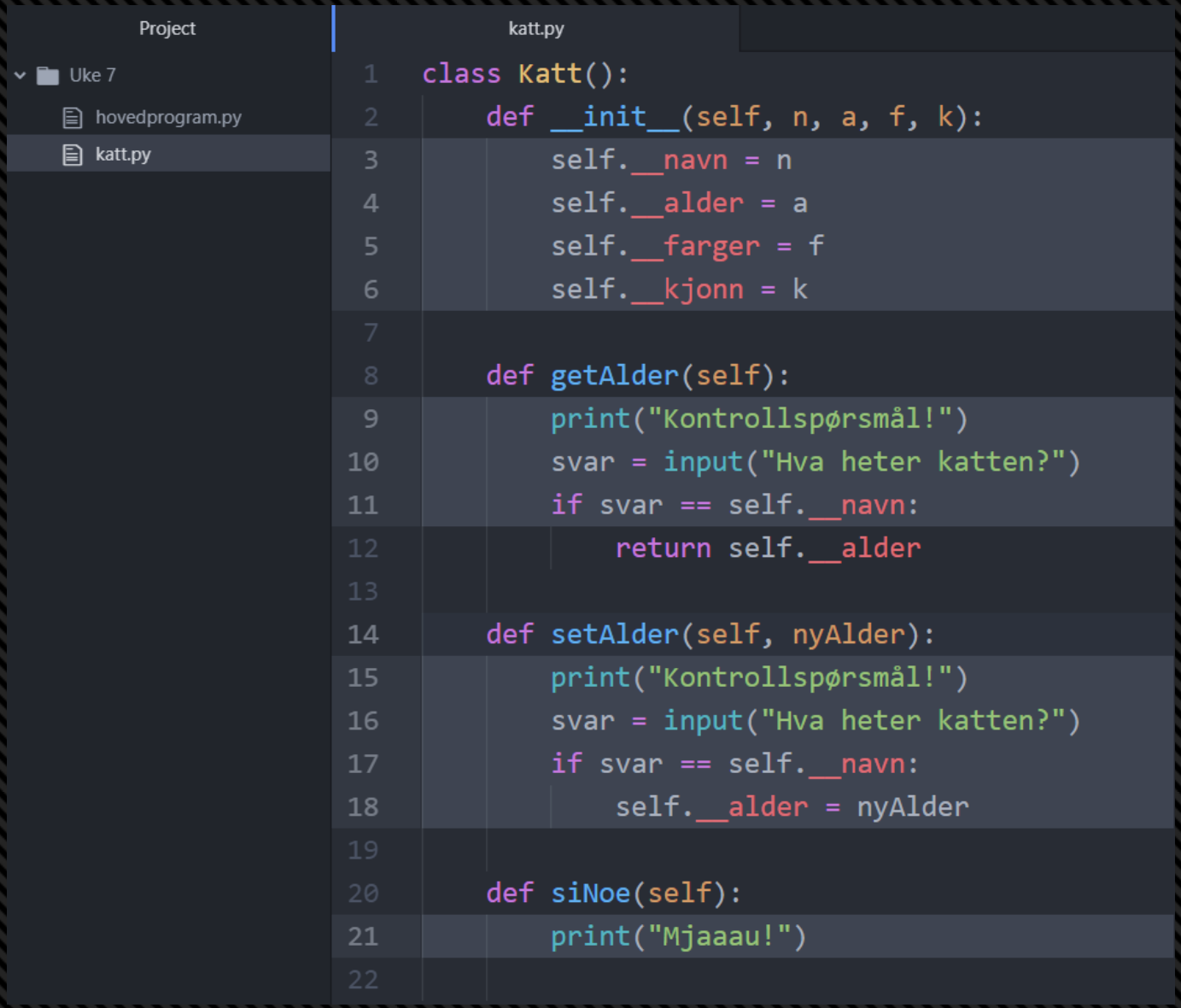
- Alle klasser har et grensesnitt (interface) og en implementasjon (implementation)
- Grensesnittet består at navnet, parameterne, og returtypen til alle prosedyrene og funksjonene dvs. alle metodene i en klasse
- Implementasjonen består av kodeblokkene som er innenfor skopet til alle prosedyrene og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

← Grensesnitt

Implementasjon →




Project
Uke 7
hovedprogram.py
katt.py
katt.py
class Katt():
def __init__(self, n, a, f, k):
self.__navn = n
self.__alder = a
self.__farger = f
self.__kjonn = k
def getAlder(self):
print("Kontrollspørsmål!")
svar = input("Hva heter katten?")
if svar == self.__navn:
return self.__alder
def setAlder(self, nyAlder):
print("Kontrollspørsmål!")
svar = input("Hva heter katten?")
if svar == self.__navn:
self.__alder = nyAlder
def siNoe(self):
print("Mjaaau!")


Project
Uke 7
hovedprogram.py
katt.py
katt.py
class Katt():
def __init__(self, n, a, f, k):
self.__navn = n
self.__alder = a
self.__farger = f
self.__kjonn = k
def getAlder(self):
print("Kontrollspørsmål!")
svar = input("Hva heter katten?")
if svar == self.__navn:
return self.__alder
def setAlder(self, nyAlder):
print("Kontrollspørsmål!")
svar = input("Hva heter katten?")
if svar == self.__navn:
self.__alder = nyAlder
def siNoe(self):
print("Mjaaau!")

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.

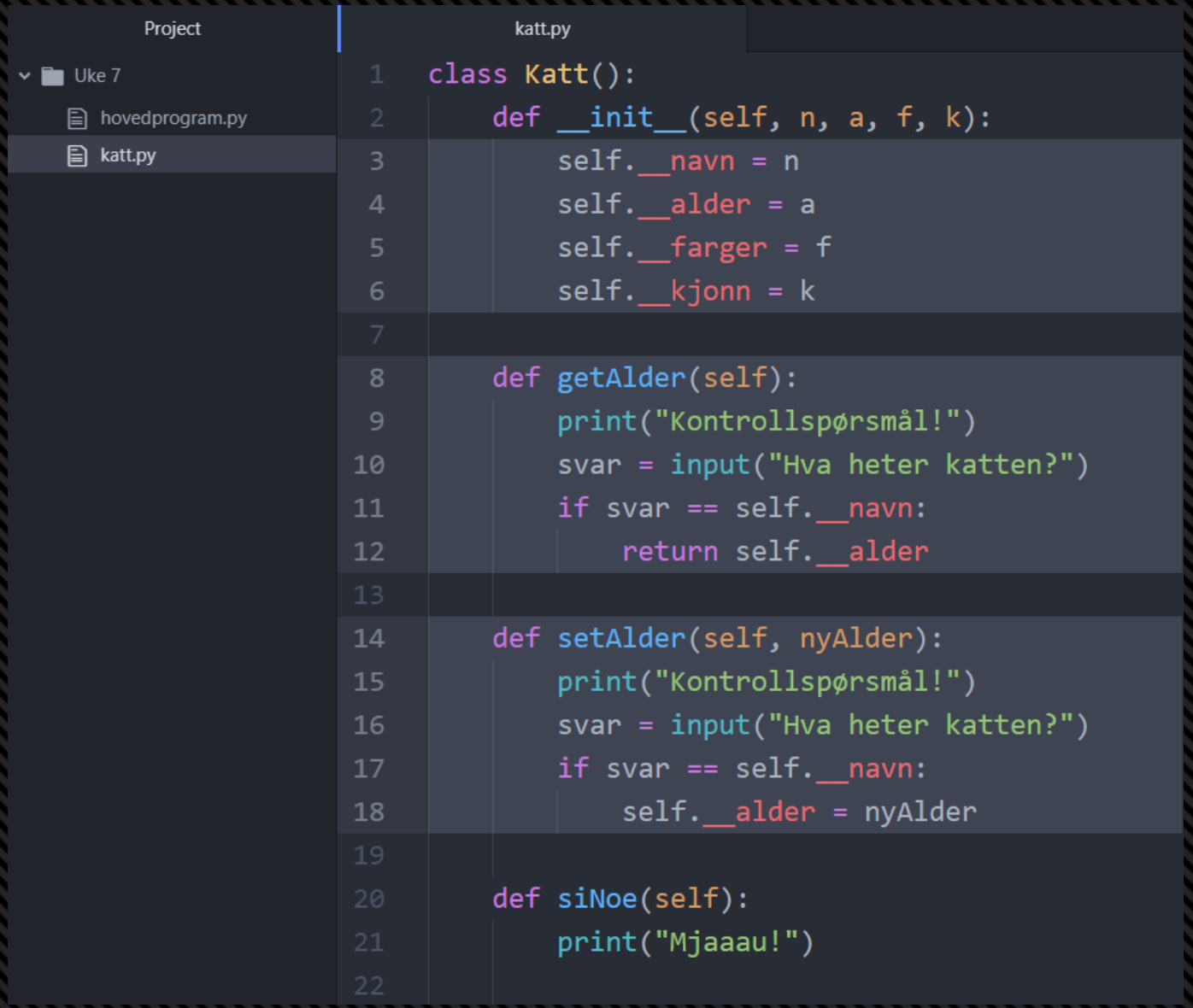
Innkapsling versus grensesnitt/implementasjon

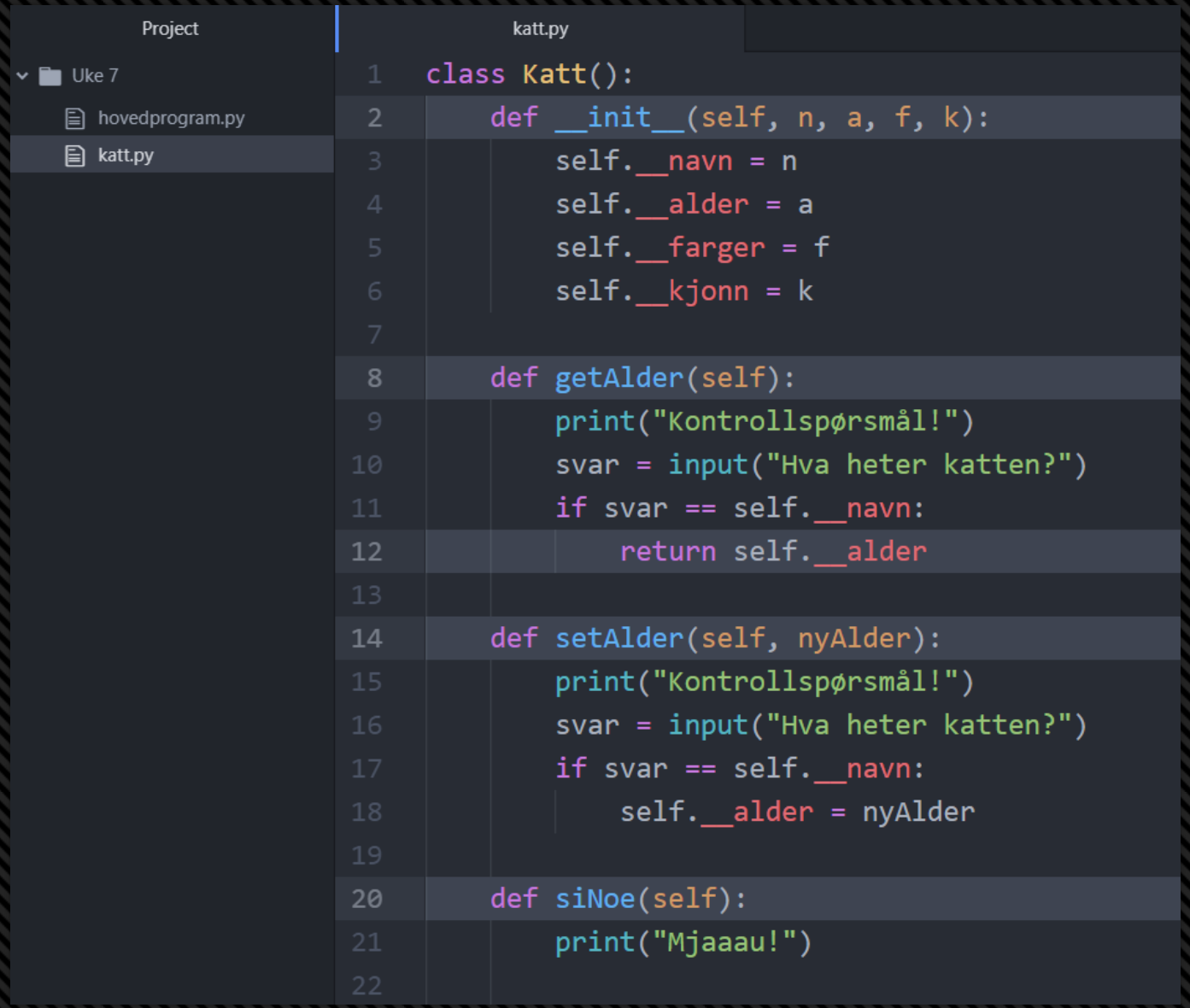
- Grensesnittet til en klasse er hvilke metoder den inneholder, hvilke parameter de har, og hva de returnerer.
- Grensesnittet er med andre ord de kodelinjene som starter på def og de kodelinjene som starter på return i en klasse.
- Implementasjonen til en klasse er hvordan metodene blir implementert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

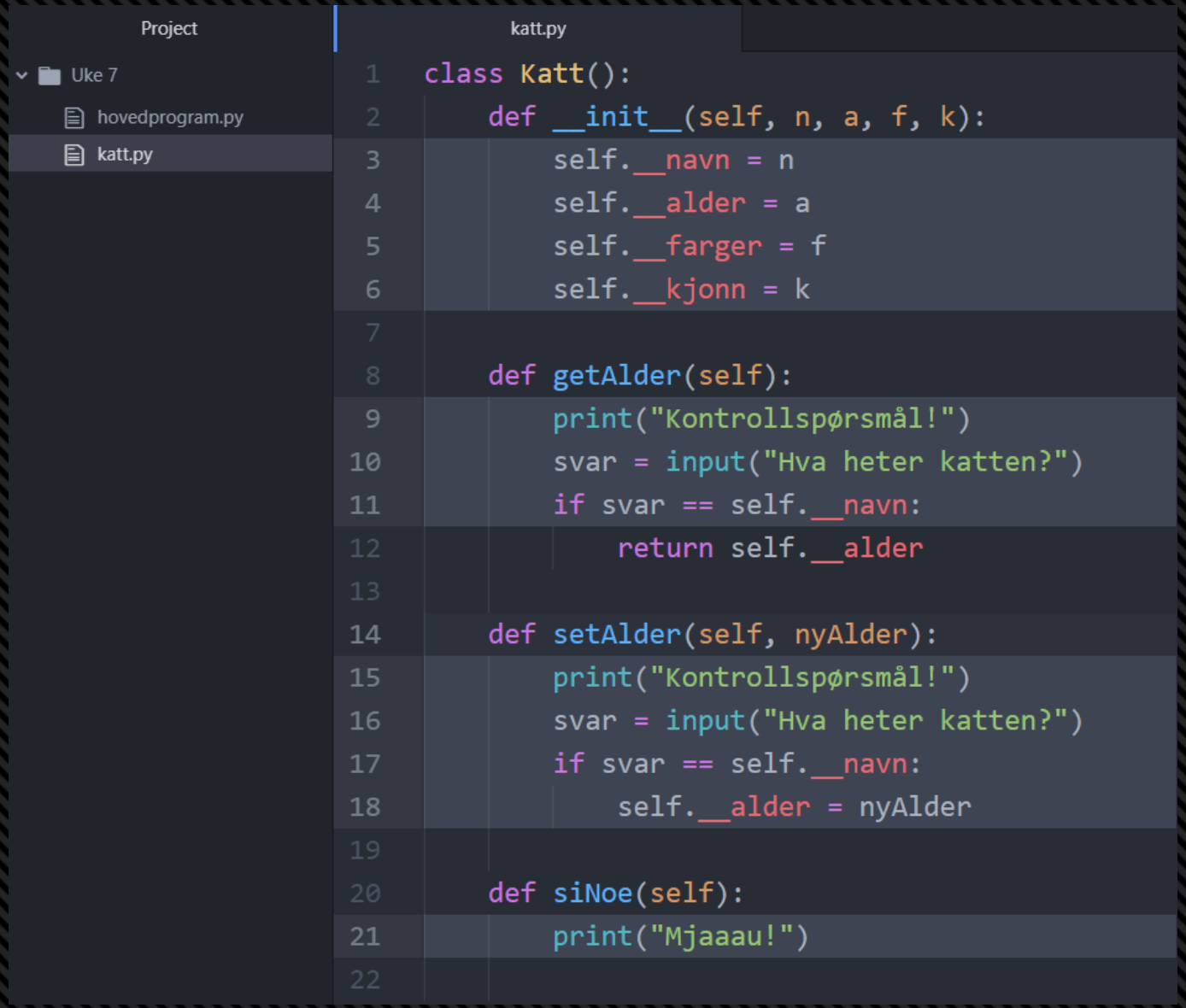
Innkapsling versus grensesnitt/implementasjon

- 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 en konstruktør eller egne metoder for å aksessere (getters) eller endre (setters) de.
- Her er en klasse med grensesnitt og implementasjon 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
class Katt():
def __init__(self, n, a, f, k):
self.__navn = n
self.__alder = a
self.__farger = f
self.__kjonn = k
def getAlder(self):
print("Kontrollspørsmål!")
svar = input("Hva heter katten?")
if svar == self.__navn:
return self.__alder
def setAlder(self, nyAlder):
print("Kontrollspørsmål!")
svar = input("Hva heter katten?")
if svar == self.__navn:
self.__alder = nyAlder
def siNoe(self):
print("Mjaaau!")


Project
Uke 7
hovedprogram.py
katt.py
katt.py
class Katt():
def __init__(self, n, a, f, k):
self.__navn = n
self.__alder = a
self.__farger = f
self.__kjonn = k
def getAlder(self):
print("Kontrollspørsmål!")
svar = input("Hva heter katten?")
if svar == self.__navn:
return self.__alder
def setAlder(self, nyAlder):
print("Kontrollspørsmål!")
svar = input("Hva heter katten?")
if svar == self.__navn:
self.__alder = nyAlder
def siNoe(self):
print("Mjaaau!")


Project
Uke 7
hovedprogram.py
katt.py
katt.py
class Katt():
def __init__(self, n, a, f, k):
self.__navn = n
self.__alder = a
self.__farger = f
self.__kjonn = k
def getAlder(self):
print("Kontrollspørsmål!")
svar = input("Hva heter katten?")
if svar == self.__navn:
return self.__alder
def setAlder(self, nyAlder):
print("Kontrollspørsmål!")
svar = input("Hva heter katten?")
if svar == self.__navn:
self.__alder = nyAlder
def siNoe(self):
print("Mjaaau!")

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 "Mjaaau!"
```