Ch.7: Innføring i klasser (del 2)

Ole Christian Lingjærde, Institutt for Informatikk, UiO

18. oktober 2018

Dagens agenda

- Lekseprøve om klasser
- Gjennomgang av prøven m/oppsummering av klasser
- Oppgave 5.16, 5.18 (evt + flere)

Bruk av klasser - lekseprøve IN1900

Oppgave 1

Hvilke av disse påstandene mener du er korrekte?

a) Moduler og klasser kan begge brukes til å samle funksjoner i en enhet.
b) Vi kan lage mange objekter (instanser) av en modul.
c) Vi kan lage mange objekter (instanser) av en klasse.
d) Alle klasser må inneholde en metode som heter **_ _init_ _**
e) Alle klasser må inneholde en metode som heter **_ _ call _ _**
f) Alle metoder i en klasse må ha **self** som første argument
g) Hvis **K** er en klasse og vi utfører setningen **p = K()** så vil metoden **_ _ init _ _** bli automatisk kalt på (utført) hvis den finnes.
h) Hvis **K** er en klasse og vi utfører setningene **p = K(); print(p)** så vil metodene **_ _ init _ _** og **_ _ call _ _** begge bli kalt på hvis de finnes.
i) Hvis vi i en metode i en klasse gjør en tilordning **x = y** så legges verdien til **y** inn i en klassevariabel (instansvariabel) **x.**
j) (Vanskelig) Hvis klassen **K** har en metode **f = lambda self: self** og vi utfører **p = K()** så vil uttrykket **id(p) == id(p.f())** få verdien **True**.

Oppgave 2

Hva blir skrevet ut her?

```
class Person:
    def __init__(self, navn, fnr, adresse):
        self.navn = navn
        self.fnr = fnr
        self.adresse = adresse

    def endreAdresse(self, nyadresse):
        self.adresse = nyadresse

    def hentAdresse(self):
        return self.adresse

    def dump(self):
        s1 = 'Navn: %s' % self.navn
        s2 = 'Fnr: %s' % self.fnr
        s3 = 'Adresse: %s' % self.adresse
        s = '; '.join([s1,s2,s3])
        print(s)

p1 = Person('Arne Arnesen', '12050112345', 'Storgata 4, Oslo')
p2 = Person('Ole Olsen', '11040023456', 'Lillegata 8, Bergen')
p1.endreAdresse(p2.hentAdresse())
p1.dump()
```

Oppgave 3

Du får programmet nedenfor i hendene - forklar i detalj hva det er som skjer når programmet kjøres - og sett opp relevant matematisk funksjon.

```
from math import exp
import matplotlib.pyplot as plt

class Weibull:
    def __init__(self, lam, k):
        self.lam = lam
        self.k = k

    def __call__(self, x):
        if x >= 0:
            u = self.k/self.lam
            v = x/self.lam
            k = self.k
            ans = u * v**(k-1) * exp(-v**k)
        else:
            ans = 0
        return ans

f = Weibull(1, 0.5)
xval = [i/100.0 for i in range(1,250)]
yval = [f(x) for x in xval]
plt.plot(xval, yval)
```

Oppgave 4

Implementer følgende funksjon (med én variabel og én parameter) i Python ved hjelp av klasser (den greske bokstaven uttales "ksi"):

$$F(s;\xi) = \begin{cases} \exp(-(1+\xi s)^{-1/\xi}) & \xi \neq 0 \\ \exp(-\exp(-s)) & \xi = 0 \end{cases}$$

Oppgave 5

Finn feilene i følgende program som implementerer en sirkel:

```
class Circle
    def init(x0, y0, R):
        self.x0, self.y0, self.R = x0, y0, R

    def area():
        return 2 * pi * self.R**2

    def circumference():
        return 2 * pi * R

c = Circle()
print(c.area(self))
```

Svar på oppgave 1

a) Korrekt
b) Feil
c) Korrekt
d) Feil (men det er vanlig)
e) Feil (men vanlig når klassen implementerer en funksjon)
f) Korrekt
g) Korrekt
h) Korrekt
i) Feil (da måtte vi skrevet `self.x = y`)
j) Korrekt (kallet `p.f()` returnerer peker til objektet)

Svar på oppgave 1

a) Korrekt
b) Feil
c) Korrekt
d) Feil (men det er vanlig)
e) Feil (men vanlig når klassen implementerer en funksjon)
f) Korrekt
g) Korrekt
h) Korrekt
i) Feil (da måtte vi skrevet `self.x = y`)
j) Korrekt (kallet `p.f()` returnerer peker til objektet)

Svar på oppgave 1

a) Korrekt
b) Feil
c) Korrekt
d) Feil (men det er vanlig)
e) Feil (men vanlig når klassen implementerer en funksjon)
f) Korrekt
g) Korrekt
h) Korrekt
i) Feil (da måtte vi skrevet `self.x = y`)
j) Korrekt (kallet `p.f()` returnerer peker til objektet)

Svar på oppgave 1

a) Korrekt
b) Feil
c) Korrekt
d) Feil (men det er vanlig)
e) Feil (men vanlig når klassen implementerer en funksjon)
f) Korrekt
g) Korrekt
h) Korrekt
i) Feil (da måtte vi skrevet `self.x = y`)
j) Korrekt (kallet `p.f()` returnerer peker til objektet)

Svar på oppgave 1

a) Korrekt
b) Feil
c) Korrekt
d) Feil (men det er vanlig)
e) Feil (men vanlig når klassen implementerer en funksjon)
f) Korrekt
g) Korrekt
h) Korrekt
i) Feil (da måtte vi skrevet `self.x = y`)
j) Korrekt (kallet `p.f()` returnerer peker til objektet)

Svar på oppgave 1

a) Korrekt
b) Feil
c) Korrekt
d) Feil (men det er vanlig)
e) Feil (men vanlig når klassen implementerer en funksjon)
f) Korrekt
g) Korrekt
h) Korrekt
i) Feil (da måtte vi skrevet `self.x = y`)
j) Korrekt (kallet `p.f()` returnerer peker til objektet)

Svar på oppgave 1

a) Korrekt
b) Feil
c) Korrekt
d) Feil (men det er vanlig)
e) Feil (men vanlig når klassen implementerer en funksjon)
f) Korrekt
g) Korrekt
h) Korrekt
i) Feil (da måtte vi skrevet `self.x = y`)
j) Korrekt (kallet `p.f()` returnerer peker til objektet)

Svar på oppgave 1

a) Korrekt
b) Feil
c) Korrekt
d) Feil (men det er vanlig)
e) Feil (men vanlig når klassen implementerer en funksjon)
f) Korrekt
g) Korrekt
h) Korrekt
i) Feil (da måtte vi skrevet `self.x = y`)
j) Korrekt (kallet `p.f()` returnerer peker til objektet)

Svar på oppgave 1

a) Korrekt
b) Feil
c) Korrekt
d) Feil (men det er vanlig)
e) Feil (men vanlig når klassen implementerer en funksjon)
f) Korrekt
g) Korrekt
h) Korrekt
i) Feil (da måtte vi skrevet `self.x = y`)
j) Korrekt (kallet `p.f()` returnerer peker til objektet)

Svar på oppgave 1

a) Korrekt
b) Feil
c) Korrekt
d) Feil (men det er vanlig)
e) Feil (men vanlig når klassen implementerer en funksjon)
f) Korrekt
g) Korrekt
h) Korrekt
i) Feil (da måtte vi skrevet `self.x = y`)
j) Korrekt (kallet `p.f()` returnerer peker til objektet)

Svar på oppgave 1

a) Korrekt
b) Feil
c) Korrekt
d) Feil (men det er vanlig)
e) Feil (men vanlig når klassen implementerer en funksjon)
f) Korrekt
g) Korrekt
h) Korrekt
i) Feil (da måtte vi skrevet `self.x = y`)
j) Korrekt (kallet `p.f()` returnerer peker til objektet)

Svar på oppgave 2

- Setningene `p1 = Person(...)` og `p2 = Person(...)` oppretter to objekter av klassen `Person` og legger argumentverdier inn i klassevariablene `self.navn`, `self.fnr` og `self.adresse`.
- Setningen `p1.endreAdresse(p2.hentAdresse())` henter adressen til Ole Olsen og legger den inn i objektet til Arne Arnesen.
- Setningen `p1.dump()` skriver ut innholdet av objektet til Arne Arnesen.
- Utskriften blir dermed som vist nedenfor.

```
Navn: Arne Arnesen; Fnr: 1205011234 5; Adresse: Lillegata 8, Bergen
```

Svar på oppgave 2

- Setningene `p1 = Person(...)` og `p2 = Person(...)` oppretter to objekter av klassen `Person` og legger argumentverdier inn i klassevariablene `self.navn`, `self.fnr` og `self.adresse`.
- Setningen `p1.endreAdresse(p2.hentAdresse())` henter adressen til Ole Olsen og legger den inn i objektet til Arne Arnesen.
- Setningen `p1.dump()` skriver ut innholdet av objektet til Arne Arnesen.
- Utskriften blir dermed som vist nedenfor.

```
Navn: Arne Arnesen; Fnr: 1205011234 5; Adresse: Lillegata 8, Bergen
```

Svar på oppgave 2

- Setningene `p1 = Person(...)` og `p2 = Person(...)` oppretter to objekter av klassen `Person` og legger argumentverdier inn i klassevariablene `self.navn`, `self.fnr` og `self.adresse`.
- Setningen `p1.endreAdresse(p2.hentAdresse())` henter adressen til Ole Olsen og legger den inn i objektet til Arne Arnesen.
- Setningen `p1.dump()` skriver ut innholdet av objektet til Arne Arnesen.
- Utskriften blir dermed som vist nedenfor.

```
Navn: Arne Arnesen; Fnr: 1205012345; Adresse: Lillegata 8, Bergen
```

Svar på oppgave 2

- Setningene `p1 = Person(...)` og `p2 = Person(...)` oppretter to objekter av klassen `Person` og legger argumentverdier inn i klassevariablene `self.navn`, `self.fnr` og `self.adresse`.
- Setningen `p1.endreAdresse(p2.hentAdresse())` henter adressen til Ole Olsen og legger den inn i objektet til Arne Arnesen.
- Setningen `p1.dump()` skriver ut innholdet av objektet til Arne Arnesen.
- Utskriften blir dermed som vist nedenfor.

```
Navn: Arne Arnesen; Fnr: 12050112345; Adresse: Lillegata 8, Bergen
```

Svar på oppgave 2

- Setningene `p1 = Person(...)` og `p2 = Person(...)` oppretter to objekter av klassen `Person` og legger argumentverdier inn i klassevariablene `self.navn`, `self.fnr` og `self.adresse`.
- Setningen `p1.endreAdresse(p2.hentAdresse())` henter adressen til Ole Olsen og legger den inn i objektet til Arne Arnesen.
- Setningen `p1.dump()` skriver ut innholdet av objektet til Arne Arnesen.
- Utskriften blir dermed som vist nedenfor.

```
Navn: Arne Arnesen; Fnr: 12050112345; Adresse: Lillegata 8, Bergen
```

Svar på oppgave 2

- Setningene `p1 = Person(...)` og `p2 = Person(...)` oppretter to objekter av klassen `Person` og legger argumentverdier inn i klassevariablene `self.navn`, `self.fnr` og `self.adresse`.
- Setningen `p1.endreAdresse(p2.hentAdresse())` henter adressen til Ole Olsen og legger den inn i objektet til Arne Arnesen.
- Setningen `p1.dump()` skriver ut innholdet av objektet til Arne Arnesen.
- Utskriften blir dermed som vist nedenfor.

```
Navn: Arne Arnesen; Fnr: 12050112345; Adresse: Lillegata 8, Bergen
```

Svar på oppgave 3

- Importerer funksjonen `exp` fra modulen `math`, og plottemodulen `matplotlib.pyplot` som `plt`
- Definerer klassen `Weibull` med to attributter (`self.lam` og `self.k`) og to metoder (`__init__` og `__call__`).
- Konstruktøren setter verdiene til de to attributtene.
- Metoden `__call__` implementerer funksjonen

$$f(x;\lambda,k) = \begin{cases} \frac{k}{\lambda}\left(\frac{x}{\lambda}\right)^{k-1} e^{-(x/\lambda)^k}, & x \geq 0 \\ 0, & x < 0 \end{cases}$$

- `f = Weibull(1, 0.5)` oppretter et objekt av `Weibull` med `self.lam=1` og `self.k=0.5` og `f` peker på objektet.
- Lager uniformt grid på intervallet $[0.01, 2.49]$.
- Regner ut $f(x;\lambda,k)$ for alle x i gridet og $\lambda = 1$, $k = 0.5$.
- Plotter $f(x;\lambda,k)$ som funksjon av x, for $\lambda = 1$, $k = 0.5$.

Svar på oppgave 3

- Importerer funksjonen `exp` fra modulen `math`, og plottemodulen `matplotlib.pyplot` som `plt`
- Definerer klassen `Weibull` med to attributter (`self.lam` og `self.k`) og to metoder (`__init__` og `__call__`).
- Konstruktøren setter verdiene til de to attributtene.
- Metoden `__call__` implementerer funksjonen

$$f(x;\lambda,k) = \begin{cases} \frac{k}{\lambda}\left(\frac{x}{\lambda}\right)^{k-1} e^{-(x/\lambda)^k}, & x \geq 0 \\ 0, & x < 0 \end{cases}$$

- `f = Weibull(1, 0.5)` oppretter et objekt av `Weibull` med `self.lam=1` og `self.k=0.5` og `f` peker på objektet.
- Lager uniformt grid på intervallet $[0.01, 2.49]$.
- Regner ut $f(x;\lambda,k)$ for alle x i gridet og $\lambda = 1$, $k = 0.5$.
- Plotter $f(x;\lambda,k)$ som funksjon av x, for $\lambda = 1$, $k = 0.5$.

Svar på oppgave 3

- Importerer funksjonen `exp` fra modulen `math`, og plottemodulen `matplotlib.pyplot` som `plt`
- Definerer klassen `Weibull` med to attributter (`self.lam` og `self.k`) og to metoder (`__init__` og `__call__`).
- Konstruktøren setter verdiene til de to attributtene.
- Metoden `__call__` implementerer funksjonen

$$f(x;\lambda,k) = \begin{cases} \frac{k}{\lambda}\left(\frac{x}{\lambda}\right)^{k-1} e^{-(x/\lambda)^k}, & x \geq 0 \\ 0, & x < 0 \end{cases}$$

- `f = Weibull(1, 0.5)` oppretter et objekt av `Weibull` med `self.lam=1` og `self.k=0.5` og `f` peker på objektet.
- Lager uniformt grid på intervallet $[0.01, 2.49]$.
- Regner ut $f(x;\lambda,k)$ for alle x i gridet og $\lambda = 1$, $k = 0.5$.
- Plotter $f(x;\lambda,k)$ som funksjon av x, for $\lambda = 1$, $k = 0.5$.

Svar på oppgave 3

- Importerer funksjonen `exp` fra modulen `math`, og plottemodulen `matplotlib.pyplot` som `plt`
- Definerer klassen `Weibull` med to attributter (`self.lam` og `self.k`) og to metoder (`__init__` og `__call__`).
- Konstruktøren setter verdiene til de to attributtene.
- Metoden `__call__` implementerer funksjonen

$$f(x;\lambda,k) = \begin{cases} \frac{k}{\lambda}\left(\frac{x}{\lambda}\right)^{k-1} e^{-(x/\lambda)^k}, & x \geq 0 \\ 0, & x < 0 \end{cases}$$

- `f = Weibull(1, 0.5)` oppretter et objekt av `Weibull` med `self.lam=1` og `self.k=0.5` og `f` peker på objektet.
- Lager uniformt grid på intervallet $[0.01, 2.49]$.
- Regner ut $f(x;\lambda,k)$ for alle x i gridet og $\lambda = 1$, $k = 0.5$.
- Plotter $f(x;\lambda,k)$ som funksjon av x, for $\lambda = 1$, $k = 0.5$.

Svar på oppgave 3

- Importerer funksjonen `exp` fra modulen `math`, og plottemodulen `matplotlib.pyplot` som `plt`
- Definerer klassen `Weibull` med to attributter (`self.lam` og `self.k`) og to metoder (`__init__` og `__call__`).
- Konstruktøren setter verdiene til de to attributtene.
- Metoden `__call__` implementerer funksjonen

$$f(x;\lambda,k) = \begin{cases} \frac{k}{\lambda}\left(\frac{x}{\lambda}\right)^{k-1} e^{-(x/\lambda)^k}, & x \geq 0 \\ 0, & x < 0 \end{cases}$$

- `f = Weibull(1, 0.5)` oppretter et objekt av `Weibull` med `self.lam=1` og `self.k=0.5` og `f` peker på objektet.
- Lager uniformt grid på intervallet $[0.01, 2.49]$.
- Regner ut $f(x;\lambda,k)$ for alle x i gridet og $\lambda = 1$, $k = 0.5$.
- Plotter $f(x;\lambda,k)$ som funksjon av x, for $\lambda = 1$, $k = 0.5$.

Svar på oppgave 3

- Importerer funksjonen `exp` fra modulen `math`, og plottemodulen `matplotlib.pyplot` som `plt`
- Definerer klassen `Weibull` med to attributter (`self.lam` og `self.k`) og to metoder (`__init__` og `__call__`).
- Konstruktøren setter verdiene til de to attributtene.
- Metoden `__call__` implementerer funksjonen

$$f(x;\lambda,k) = \begin{cases} \frac{k}{\lambda}\left(\frac{x}{\lambda}\right)^{k-1} e^{-(x/\lambda)^k}, & x \geq 0 \\ 0, & x < 0 \end{cases}$$

- `f = Weibull(1, 0.5)` oppretter et objekt av `Weibull` med `self.lam=1` og `self.k=0.5` og `f` peker på objektet.
- Lager uniformt grid på intervallet $[0.01, 2.49]$.
- Regner ut $f(x;\lambda,k)$ for alle x i gridet og $\lambda = 1$, $k = 0.5$.
- Plotter $f(x;\lambda,k)$ som funksjon av x, for $\lambda = 1$, $k = 0.5$.

Svar på oppgave 3

- Importerer funksjonen `exp` fra modulen `math`, og plottemodulen `matplotlib.pyplot` som `plt`
- Definerer klassen `Weibull` med to attributter (`self.lam` og `self.k`) og to metoder (`__init__` og `__call__`).
- Konstruktøren setter verdiene til de to attributtene.
- Metoden `__call__` implementerer funksjonen

$$f(x;\lambda,k) = \begin{cases} \frac{k}{\lambda}\left(\frac{x}{\lambda}\right)^{k-1} e^{-(x/\lambda)^k}, & x \geq 0 \\ 0, & x < 0 \end{cases}$$

- `f = Weibull(1, 0.5)` oppretter et objekt av `Weibull` med `self.lam=1` og `self.k=0.5` og `f` peker på objektet.
- Lager uniformt grid på intervallet $[0.01, 2.49]$.
- Regner ut $f(x;\lambda,k)$ for alle x i gridet og $\lambda = 1$, $k = 0.5$.
- Plotter $f(x;\lambda,k)$ som funksjon av x, for $\lambda = 1$, $k = 0.5$.

Svar på oppgave 3

- Importerer funksjonen `exp` fra modulen `math`, og plottemodulen `matplotlib.pyplot` som `plt`
- Definerer klassen `Weibull` med to attributter (`self.lam` og `self.k`) og to metoder (`__init__` og `__call__`).
- Konstruktøren setter verdiene til de to attributtene.
- Metoden `__call__` implementerer funksjonen

$$f(x;\lambda,k) = \begin{cases} \frac{k}{\lambda}\left(\frac{x}{\lambda}\right)^{k-1} e^{-(x/\lambda)^k}, & x \geq 0 \\ 0, & x < 0 \end{cases}$$

- `f = Weibull(1, 0.5)` oppretter et objekt av `Weibull` med `self.lam=1` og `self.k=0.5` og `f` peker på objektet.
- Lager uniformt grid på intervallet $[0.01, 2.49]$.
- Regner ut $f(x;\lambda,k)$ for alle x i gridet og $\lambda = 1$, $k = 0.5$.
- Plotter $f(x;\lambda,k)$ som funksjon av x, for $\lambda = 1$, $k = 0.5$.

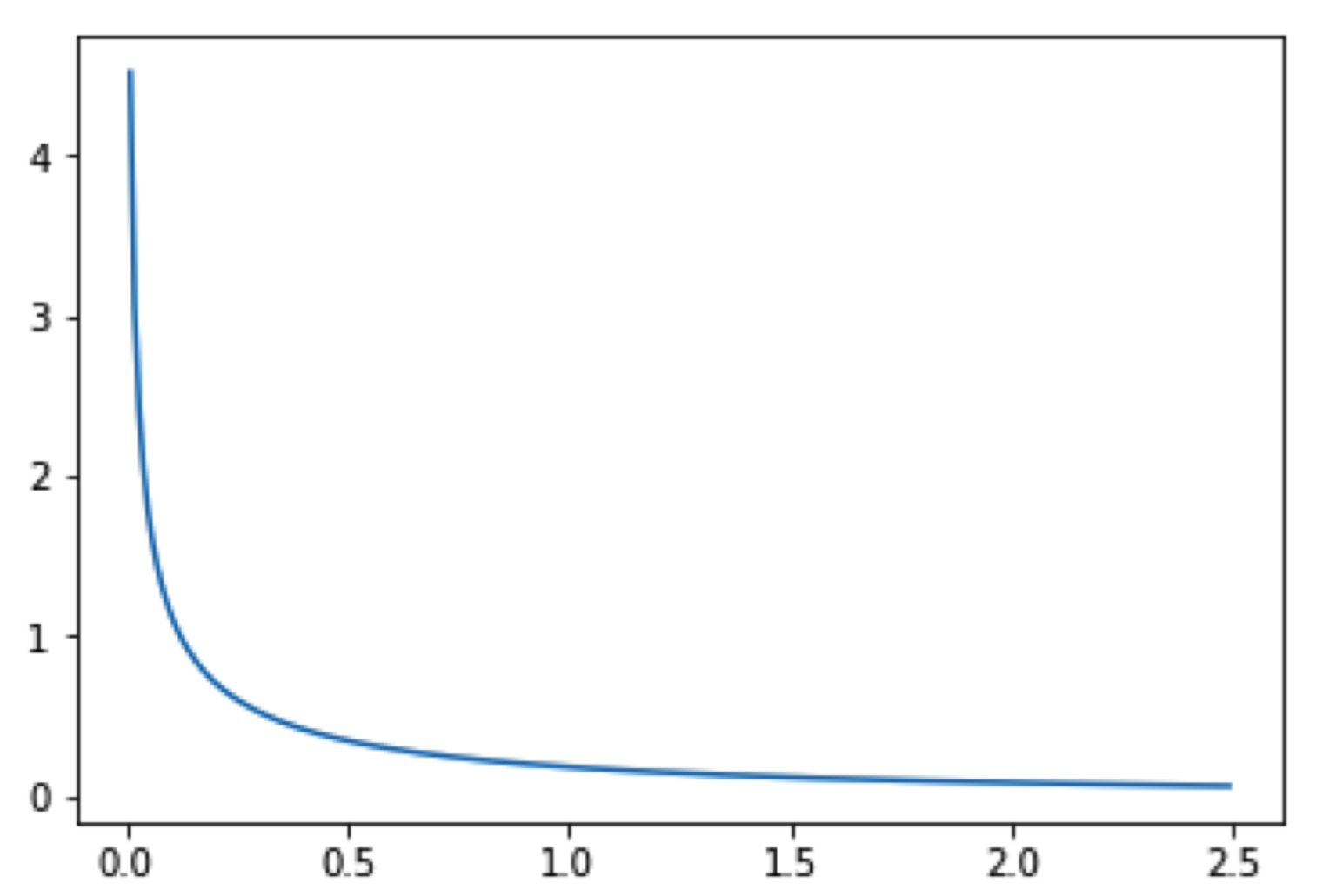
Svar på oppgave 3

- Importerer funksjonen `exp` fra modulen `math`, og plottemodulen `matplotlib.pyplot` som `plt`
- Definerer klassen `Weibull` med to attributter (`self.lam` og `self.k`) og to metoder (`__init__` og `__call__`).
- Konstruktøren setter verdiene til de to attributtene.
- Metoden `__call__` implementerer funksjonen

$$f(x;\lambda,k) = \begin{cases} \frac{k}{\lambda}\left(\frac{x}{\lambda}\right)^{k-1} e^{-(x/\lambda)^k}, & x \geq 0 \\ 0, & x < 0 \end{cases}$$

- `f = Weibull(1, 0.5)` oppretter et objekt av `Weibull` med `self.lam=1` og `self.k=0.5` og `f` peker på objektet.
- Lager uniformt grid på intervallet $[0.01, 2.49]$.
- Regner ut $f(x;\lambda,k)$ for alle x i gridet og $\lambda = 1$, $k = 0.5$.
- Plotter $f(x;\lambda,k)$ som funksjon av x, for $\lambda = 1$, $k = 0.5$.

Resultat



Svar på oppgave 4

```
from math import exp

class FClass:
    def __init__(self, ksi):
        self.ksi = ksi

    def __call__(self, s):
        ksi = self.ksi
        if ksi != 0:
            ans = exp(-(1+ksi*s)**(-1.0/ksi))
        else:
            ans = exp(-exp(-s))
        return ans

# Test of function
F = FClass(0.5)
print(F(0.5))
```

Svar på oppgave 5

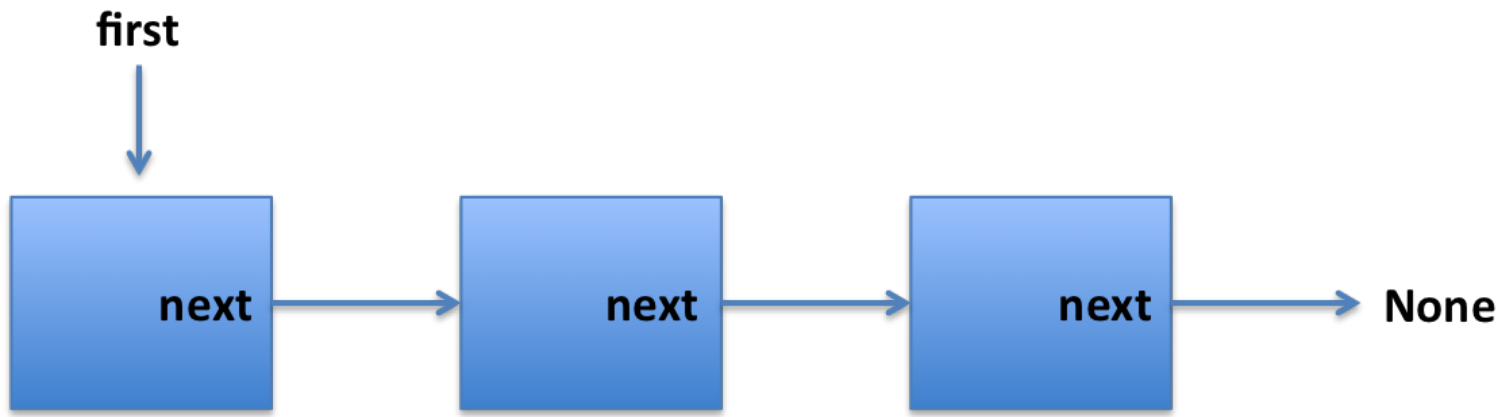
```
class Circle                          # Mangler kolon på slutten
    def init(x0, y0, R):              # Mangler self
        self.x0, self.y0, self.R = x0, y0, R

    def area():                               # Mangler self
        return 2 * pi * self.R**2             # pi ikke definert

    def circumference():                      # Mangler self
        return 2 * pi * R                     # pi ikke definert,
                                              # R skulle vært self.R

c = Circle()                          # Mangler tre argumenter
print(c.area(self))                   # Skal ikke ha med self
```

I tillegg er formelen for arealet av en sirkel feil.

Eksempel: Enveis lenkede lister

Tenk deg at vi har laget mange objekter av en klasse (f.eks. `class Person`). Vi har en peker til hvert objekt og ønsker å lagre alle objektene samlet. Hvordan gjør vi det?

- En åpenbar løsning er å legge alle pekerne inn i en liste.
- En annen metode er å bare ha en peker til det første elementet i listen, og så overlate til hvert objekt å holde rede på hva neste element i listen er:



Mulig implementasjon

```
# Define class Person
class Person:
    def __init__(self, navn, next):
        self.navn = navn
        self.next = next

    def __str__(self):
        return self.navn

# Create linked list
first = Person('Ole Olsen', None)
first = Person('Jens Jensen', first)
first = Person('Kari Olsen', first)
first = Person('Unn Olsen', first)

# Go through list and print names
p = first
while (p is not None):
    print(p)
    p = p.next
```

Resultat

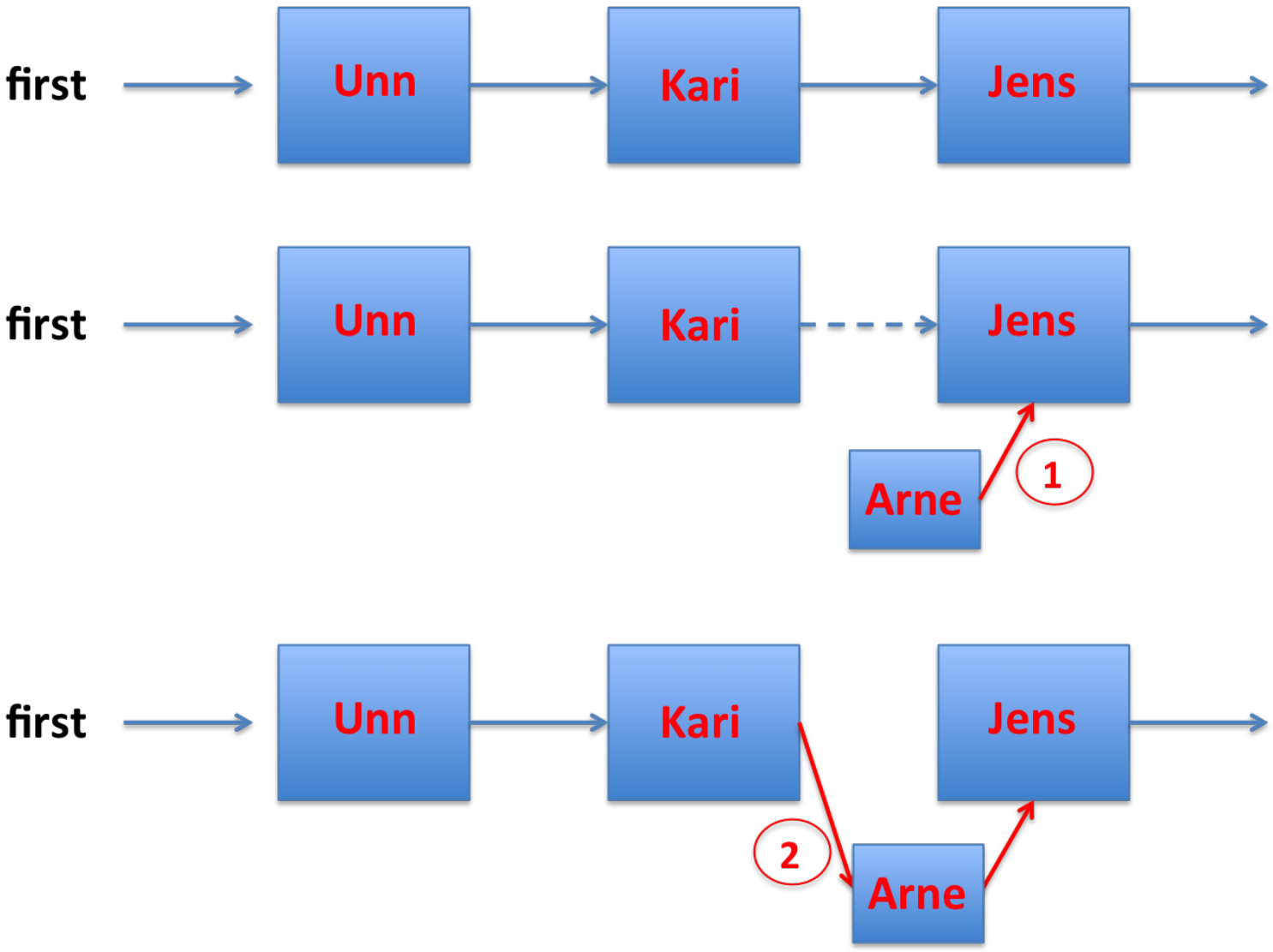
```
Unn Olsen
Kari Olsen
Jens Jensen
Ole Olsen
```

NB: merk rekkefølgen navnene skrives ut i. Det første objektet vi lager havner bakerst i listen, og det siste objektet havner fremst i listen.

Legge til element midt i listen

- Anta at `first` peker på første objekt i en enveis lenket liste som inneholder personene Unn Olsen, Kari Olsen, Jens Jensen og Ole Olsen i den rekkefølgen.
- Vi ønsker å sette inn et nytt element i listen, mellom Kari Olsen og Jens Jensen.
- Hvordan får vi til det?

Mulig løsning (visuelt)



Mulig løsning (Python-kode)

```
# Finn pekere til Kari og Jens:
kari = first.next
jens = kari.next
# Lag objekt for Arne og få det til å peke på Jens:
arne = Person('Arne Arnesen', jens)
# Få til slutt Kari til å peke på Arne
kari.next = arne
```

Merk 1: Vi måtte sette *to* pekere for å få det til.

Merk 2: Normalt ville vi ikke valgt navn på pekerne som i eksemplet ovenfor, det var kun gjort for å gjøre det lett å følge gangen i operasjonen.

Eksempel: Toveis lenkede lister

- En ulempe med enveis lenkede lister er at vi bare kan gå gjennom elementene i én retning (fra første element til siste).
- Vi kan omgå dette problemet ved å innføre *to* pekere i hvert objekt: en som peker forover og en som peker bakover.
- Lett å implementere - vi hopper over detaljene her.

Exercise 5.16

Plot data from a file

- The files `density_water.dat` and `density_air.dat` in the folder `src/plot13` contain data about the density of water and air (respectively) for different temperatures.
- The data files have some comment lines starting with hashtag and some lines are blank. The rest of the lines contain density data: the temperature in the first column and the corresponding density in the second column.
- The goal of this exercise is to read the data in such a file and plot the density versus the temperature as distinct (small) circles for each data point. Let the program take the name of the data file as command-line argument. Apply the program to both files.
- Filename: `read_density_data.`

density_water.dat

```
# Density of air at different temperatures, at 1 atm pressure
# Column 1: temperature in Celsius degrees
# Column 2: density in kg/m^3

  0.0   999.8425
  4.0   999.9750
 15.0   999.1026
 20.0   998.2071
 25.0   997.0479
 37.0   993.3316
 50.0   988.04
100.0   958.3665
# Source: Wikipedia (keyword Density)
```

density_air.dat

```
# Density of air at different temperatures, at 1 atm pressure
# Column 1: temperature in Celsius degrees
# Column 2: density in kg/m^3

-10     1.341
 -5     1.316
  0     1.293
  5     1.269
 10     1.247
 15     1.225
 20     1.204
 25     1.184
 30     1.164
# Source: Wikipedia (keyword Density)
```

Planlegging av arbeidet

- Vi henter `density_water.dat` og `density_air.dat` fra nettstedet `https://github.com/hplgit/scipro-primer/` og lagrer dem lokalt samme sted som vi skal legge programfilen vi lager.
- Vi får tak i filnavnet i programmet via `sys.argv[1]`.
- Vi leser hver fil linje-for-linje og splitter opp i enkeltord for å få tak i de to tallene på hver linje.
- For hver linje vi leser, må vi huske først å sjekke om det er en kommentar/blank linje. Isåfall gjør vi ingenting bortsett fra å fortsette å lese.
- Vi bruker pakken `matplotlib.pyplot` til å plotte.

Implementasjon

```
### Exercise 5.16

import sys
import matplotlib.pyplot as plt

fname = sys.argv[1]
temp = []; dens = []
infile = open(fname, 'r')
for line in infile:
    words = line.split();
    if len(words) >= 2 and words[0] != '#':
        temp.append(float(words[0]))
        dens.append(float(words[1]))
infile.close()
plt.plot(temp, dens, 'bo')
plt.show()
```

Exercise 5.18

Fit a polynomial to data points

The purpose of this exercise is to find a simple mathematical formula for how the density of water or air depends on the temperature.

The idea is to load density and temperature data from file as explained in Exercise 5.16 and then apply some NumPy utilities that can find a polynomial that approximates the density as a function of the temperature.

Exercise 5.18 (cont'd)

NumPy has a function `polyfit(x, y, deg)` for finding a best fit of a polynomial of degree deg to a set of data points given by the array arguments `x` and `y`. The polyfit function returns a list of the coefficients in the fitted polynomial, where the first element is the coefficient for the term with the highest degree, and the last element corresponds to the constant term.

For example, given points in `x` and `y`, `polyfit(x, y, 1)` returns the coefficients a, b in a polynomial $a * x + b$ that fits the data in the best way.

Exercise 5.18 (cont'd)

NumPy also has a utility `poly1d`, which can take the tuple or list of coefficients calculated by, e.g., `polyfit` and return the polynomial as a Python function that can be evaluated. The following code snippet demonstrates the use of `polyfit` and `poly1d`:

```
coeff = polyfit(x, y, deg)
p = poly1d(coeff)
print(p)           # Prints the polynomial expression
y_fitted = p(x)    # Computes the polynomial at the x points

# Use red circles for data points and a blue line for the polyn.
plot(x, y, 'ro', x, y_fitted, 'b-',
legend=('data', 'fitted polynomial of degree %d' % deg))
```

Exercise 5.18 (cont'd)

Questions:

- Write a function `fit(x, y, deg)` that creates a plot of data in `x` and `y` arrays along with polynomial approximations of degrees collected in the list `deg` as explained above.
- We want to call fit to make a plot of the density of water versus temperature and another plot of the density of air versus temperature. In both calls, use `deg=[1,2]` such that we can compare linear and quadratic approximations to the data.
- From a visual inspection of the plots, can you suggest simple mathematical formulas that relate the density of air to temperature and the density of water to temperature?

Filename: `fit_density_data`.

Oppgaven (kort oppsummert)

- Vi skal lage en funksjon `fit(x,y,deg)` hvor `x` og `y` er lister med datapunkter og `deg` er en liste som spesifiserer graden til polynomene vi skal bruke til å approksimere datapunktene med.
- Kodeeksemplet som er oppgitt i oppgaven viser hvordan vi finner approksimasjonene.
- Så skal vi bruke funksjonen vi laget over til å plotte noen approksimasjoner.
- Til slutt skal vi foreslå hvordan `y` og `x` er matematisk relatert.

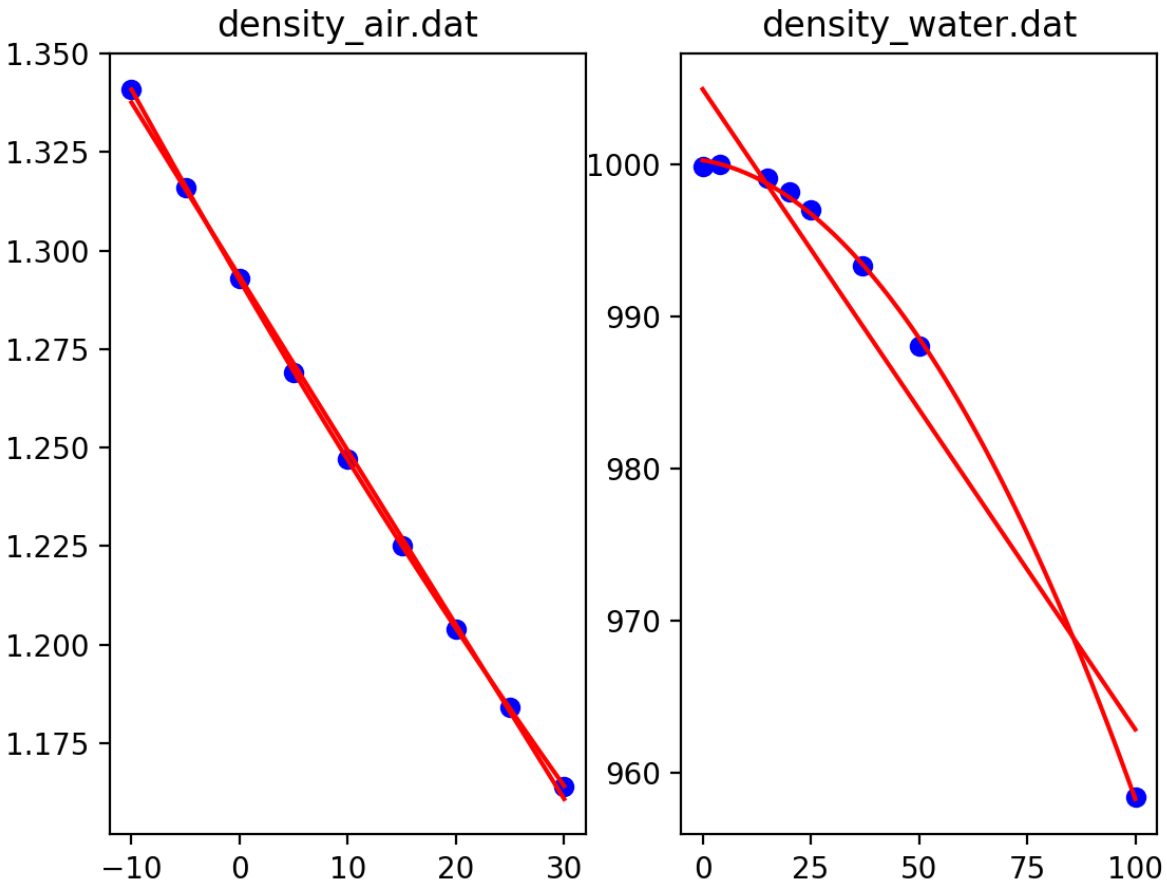
Svar på 5.18 a)

```
def fit(x, y, deg):
    import matplotlib.pyplot as plt
    import numpy as np
    plt.plot(x, y, 'bo')
    x1 = np.linspace(np.min(x), np.max(x), 100)
    for d in deg:
        coef = np.polyfit(x, y, d)  # Polynomial coefficients
        p = np.poly1d(coef)         # Polynomial function
        y1 = p(x1)                  # Polynomial value at x1 points
        plt.plot(x1, y1, 'r-')
    plt.show()
```

Svar på 5.18 b)

```
import matplotlib.pyplot as plt
import numpy as np

def fit(x, y, deg):
    plt.plot(x, y, 'bo')
    x1 = np.linspace(np.min(x), np.max(x), 100)
    for d in deg:
        coef = np.polyfit(x, y, d)  # Polynomial coefficients
        p = np.poly1d(coef)         # Polynomial function
        y1 = p(x1)                  # Polynomial value at x1 points
        plt.plot(x1, y1, 'r-')

fnames = ['density_air.dat', 'density_water.dat']

for i in range(len(fnames)):
    plt.subplot(1,2,i+1)
    temp = []; dens = []
    infile = open(fnames[i], 'r')
    for line in infile:
        words = line.split();
        if len(words) >= 2 and words[0] != '#':
            temp.append(float(words[0]))
            dens.append(float(words[1]))
    infile.close()
    fit(temp, dens, deg=[1,2])
    plt.title(fnames[i])
plt.show()
```

Resultat


density_air.dat
1.350
1.325
1.300
1.275
1.250
1.225
1.200
1.175
−10
0
10
20
30
density_water.dat
1000
990
980
970
960
0
25
50
75
100

Svar på 5.18 c)

Konklusjon

- I ett plott er den lineære og den kvadratiske tilpasningen nesten identiske, og begge tilpasser dataene godt.
- I det andre plottet er den lineære og den kvadratiske tilpasningen ganske forskjellige, og den kvadratiske tilpasningen følger dataene veldig bra.
- Vi konkluderer at i begge tilfeller er dataene konsistente med en kvadratisk relasjon mellom x = temperature og y = density.