Ch.7: Innføring i klasser

Ole Christian Lingjærde, Institutt for Informatikk, UiO

16. oktober 2018

Dagens agenda

- Hva er klasser?
- En liten quiz

Å gruppere ting i enheter

- Alle store organisasjoner er delt inn i ulike enheter.
- Eksempel: UiO har fakulteter, fakulteter har institutter, institutter har seksjoner. Dessuten sentral administrasjon, teknisk avdeling, osv.
- Hvorfor??

Hva oppnår man med å dele i enheter?

- Redusert kompleksitet
- Lettere å få oversikt over hele organisasjonen
- Klart definerte grensesnitt mellom ulike enheter
- Hver ansatt/student ser bare det de trenger å se
- Lettere å identifisere svake ledd
- Lettere å se effekten av endringer
- Mindre navneforvirring

Akkurat samme fordeler oppnår man ved å dele programmer opp i mindre enheter!

Hvordan dele programmer i enheter?

Funksjoner:
Samler flere programlinjer under et felles navn

Moduler:
Samler flere funksjoner under et felles navn

Klasser:
Samler både funksjoner og variable under et felles navn

Eksempler (uformelt)

Funksjoner:
Samle programlinjer som tilsammen utfører en beregning.
Eksempel: programlinjer som tilsammen finner et nullpunkt til en funksjon f (x slik at $f(x) = 0$).

Moduler:
Samle funksjoner som begrepsmessig hører sammen. Eksempel: `math` inneholder funksjoner for å gjøre "matte-ting" og `numpy` funksjoner for å gjøre vektoriserte numeriske beregninger.

Klasser:
Samle data som begrepsmessig hører sammen, og funksjoner som jobber på disse. Eksempel: vi kan lage en klasse `Person` som inneholder data (variable) om en enkeltperson slik som navn, kjønn, adresse, fødselsnummer, inntekt. Samt funksjoner som jobber på disse dataene, f.eks. endreAdresse. Så kan vi lage mange kopier (ofte kalt *instanser* eller *objekter*) av klassen slik at vi kan holde data om mange personer samtidig.

Eksempler (Python)

Funksjon

```
def f(x):
    res = x**2 + 2*x + 3
    return res
```

Modul

```
def f(x,a,b,c):
    return a*x**2 + b*x + c

def message(s):
    print('Message: %s' % s)
```

Klasse

```
class F:
    def __init__(self, a, b, c):
        self.a = a
        self.b = b
        self.c = c

    def __call__(self, x):
        return self.a * x**2 + self.b * x + self.c
```

Hvordan bruke dem?

Funksjon

Funksjoner i samme program kan vi bruke direkte:

```
print(f(3))
```

Modul

Vi må først importere modulen:

```
from minmodul import f
print(f(3, 0, 2, 4))
```

Klasse

Vi må først lage en instans av klassen:

```
f = F(0, 2, 4)
print(f(3))
```

Elementene i klassen F

```
class F:                                # Navnet F velger vi selv
    def __init__(self, a, b, c):       # Ble kalt her: f = F(0,2,4)
        self.a = a                      # self.a er global i klassen
        self.b = b                      # self.b tilsvarende
        self.c = c                      # self.c tilsvarende

    def __call__(self, x):              # Ble kalt her: print(f(3))
        return self.a * x**2 + self.b * x + self.c
```

Merk 1: variabelen `self` må alltid være med som første argument i alle funksjoner i klassen, men vi tar den ikke med når vi kaller på funksjonene!

Merk 2: vi kaller variablene `self.a`, `self.b`, `self.c` for attributter til klassen F.

Merk 3: vi kaller funksjonene i klassen for *metoder*.

Klasse = funksjoner + data (variable) samlet i en enhet

- En klasse pakker sammen data (= variable) og funksjoner i en enkelt enhet.
- Som programmerer kan du definere nye klasser og nye objekttyper (slik som `float`, `list`, `file`, osv)
- En klasse har mye til felles med en modul: en samling "globale" variable funksjoner som hører sammen
- MEN en modul har alltid bare én instans (kopi), mens en klasse kan ha mange instanser som "lever uavhengig av hverandre".
- Klasser er i utstrakt bruk i programmering
- Det tar litt tid å lære seg klassebegrepet ordentlig
- Umulig å lære uten trening - løs oppgaver!

Hvordan implementere en funksjon med parametre?

Anta at vi har en funksjon av t med en parameter v_0:

$$f(t; v_0) = v_0 t - \frac{1}{2} g t^2$$

Vi trenger åpenbart både t, v_0 og $g = 9.81$ for å evaluere f, men hvordan implementerer vi dette i praksis?

Tre mulige løsninger

Løsning A (Dårlig)

```
def f(t, v0, g=9.81):
    return v0*t - 0.5*g*t**2
# Typisk bruk:
value = f(1.5, 50.0)
```

Løsning B (Bedre)

```
def f(t, v0):
    g = 9.81
    return v0*t - 0.5*g*t**2
# Typisk bruk:
value = f(1.5, 50.0)
```

Løsning C (Best)

```
class F:
    def __init__(self, v0):
        self.v0 = v0
        self.g = 9.81
    def __call__(self, t):
        return self.v0*t - 0.5*self.g*t**2
# Typisk bruk:
f = F(50.0)
value = f(1.5)
```

Hva skjer i det siste eksemplet?

Når f = F(50.0) utføres::

- Det opprettes et nytt objekt (en ny *instans*) av klassen F
- Metoden `__init__(self, v0)` kalles med `v0=50.0`
- Det opprettes to nye variable i objektet: `self.v0` (som får verdi 50.0) og `self.g` (som får verdi 9.81)
- Variabelen `f` settes til å "peke på" (referere) objektet

Når value = f(1.5) utføres::

- Metoden `__call__(self, t)` kalles med `t=1.5`
- Metoden returnerer resultatverdien fra beregningen
- Variabelen `value` settes lik denne verdien

Fortsatt forvirret?

Eksemplet over er veldig forskjellig fra alt vi har sett tidligere:

- Når vi skriver `f = F(50.0)` så kalles funksjonen `__init__`
- Når vi skriver `value = f(1.5)` så kalles funksjonen `__call__`
- Hva er variabelen `self`?

For å forstå hvordan alt dette henger sammen, skal vi starte med en veldig enkel klasse og utvide i små steg!

Trinn 0

Klasse som bare inneholder variable (data):

```
class MinKlasse: # Definerer klassen "MinKlasse"
    a = 1.5      # Klassen inneholder variablene a og b
    b = 2.5

p = MinKlasse()  # Nå er p peker til et nytt objekt av MinKlasse
print(p.a)       # Skriver ut 1.5
print(p.b)       # Skriver ut 2.5
```

Viktig: Vi må skrive `p.a` for at Python skal skjønne at vi mener akkurat den variabelen `a` som ligger inni objektet som `p` peker på.

Trinn 1

Vi kan lage flere objekter (instanser) av klassen:

```
class MinKlasse:
    a = 1.5
    b = 2.5

p1 = MinKlasse()   # Lag objekt av klassen MinKlasse
p2 = MinKlasse()   # Lag et objekt til av klassen MinKlasse
p1.a = 1000        # Endre a's verdi i p2-objektet
print(p1.a)        # Skriver ut 1000
print(p2.a)        # Skriver ut 1.5
```

Merk at vi nå har totalt *fire* variable: to stykker i p1-objektet og to stykker (med samme navn) i p2-objektet!

Trinn 2

Klasse som også inneholder en funksjon:

```
class MinKlasse:
    a = 1.5
    b = 2.5

    def f(self, x):      # Argumentet self MÅ være med
        return x**2      # ...selv om vi ikke bruker det

p = MinKlasse()      # Oppretter objekt av klassen MinKlasse
print(p.f(4))        # Skriver ut 16
```

Viktig: Vi må skrive `p.f` for at Python skal skjønne at vi mener akkurat den funksjonen `f` som ligger inni objektet som `p` peker på.

Trinn 3

Vi prøver å lage flere objekter (instanser) igjen:

```
class MinKlasse:
    a = 1.5
    b = 2.5

    def f(self, x):      # Argumentet self MÅ være med
        return x**2      # ...selv om vi ikke bruker det

p1 = MinKlasse()    # Opprett objekt av klassen MinKlasse
p2 = MinKlasse()    # Opprett et objekt til av MinKlasse
print(p1.f(4))      # Skriver ut 16
print(p2.f(4))      # Skriver ut 16
```

Vi kan tenke oss at det er forskjellige funksjoner vi kaller på i de to siste setningene over: en i p1-objektet og en i p2-objektet. Men her oppfører de seg likt.

Trinn 4

Vi prøver å bruke en klassevariabel i en klassefunksjon:

```
class MinKlasse:
    a = 1.5
    b = 2.5

    def f(self, x):
        return a * x**2      # Forsøker å bruke a

p = MinKlasse()
print(p.f(4))   # FEILMELDING: global name 'a' is not defined
```

Hvorfor virker det ikke??

Svar: Når `a * x**2` evalueres, leter Python etter *lokale* variable i `f` som heter `a` og `x`. Den finner `x` men ikke `a`. Deretter leter Python etter en *global* variabel `a`. Det finner den heller ikke!

Python ignorerer fullstendig at det finnes en *klassevariabel* som heter `a`. For å bruke den, må vi skrive `self.a`. Tenke på `self` som en *intern peker* til objektet vi er inni.

Trinn 5

Korrekt forsøk på å bruke klassevariable:

```
class MinKlasse:
    a = 1.5
    b = 2.5

    def f(self, x):
        return self.a * x**2     # a byttet ut med self.a

p = MinKlasse()
print(p.f(4))                    # Nå fungerer det!
```

Trinn 6

I de foregående eksemplene er variablene a og b "hardkodet" til å være lik 1.5 og 2.5 når vi lager et nytt klasseobjekt. Det er mer fleksibelt å bestemme dem når et objekt lages. Da bruker vi en *konstruktør*:

```
class MinKlasse:
    # Funksjonen under kalles automatisk når objekt opprettes:
    def __init__(self, a, b):
        self.a = a        # Sett klassevariabel a
        self.b = b        # Sett klassevariabel b

    def f(self, x):
        return self.a * x**2    # a byttet ut med self.a

p1 = MinKlasse(1.5, 2.5)  # Lag nytt objekt, sett a=1.5 og b=2.5
p2 = MinKlasse(50, 100)   # Lag nytt objekt, sett a=50 og b=100
```

Trinn 7

Vi har sett at funksjonen `__init__` er spesiell: vi kaller aldri direkte på den, men den utføres automatisk når et nytt objekt lages.

Det finnes mange slike spesielle funksjoner som vi ikke kaller på direkte. Alle har navn av typen `__XXXX__`.

Noen få eksempler:

```
# Kalles automatisk når nytt objekt opprettes:
__init__(self, a, b)

# Kalles når p(x,y) utføres, hvor p er peker til objektet:
__call__(self, x, y)

# Kalles når print(p) utføres, hvor p er peker til objektet:
__str__(self)
```

Litt mer om variabelen `self`

- Tenk på `self` som en intern peker til objektet selv. Når en funksjon inni objektet utfører `self.v0 = ...` så blir klassevariabelen `v0` tilordnet en verdi.
- `MinKlasse(2, 3)` oppretter et nytt objekt og utfører `__init__(p, 2, 3)` hvor `p` er en peker til objektet
- Husk: `self` er alltid første parameter i klassefunksjoner, men du skal ikke oppgi den når du kaller på slike funksjoner (det gjør Python "bak kulissene").
- Etter `p = MinKlasse(2, 3)` så har objektet `p` to variable `a` og `b` som har fått tilordnet verdiene 2 og 3, henholdsvis.

Kort oppsummering så langt

- Klassen `MinKlasse` samler attributtene (variablene) `a` og `b` og metoden `f` i en enhet
- `f(self, x)` kalles med bare ett argument (`p.f(3)`), men har automatisk aksess til parametrene `self.a` og `self.b`
- Teknisk sett en stor fordel: vi kan tenke på `f` som en funksjon av bare én variabel, men den kjenner likevel (og kan bruke) verdien til variablene `self.a` og `self.b`

Generelt om funksjoner med parametre

Gitt en funksjon med $n+1$ parametre og en uavhengig variabel:

$$f(x; p_0, \ldots, p_n)$$

er det smart å bruke en klasse til å implementere `f`, hvor $p_0, \ldots, p_n$ er attributter i klassen og hvor det er en metode `f(self, x)` for å beregne $f(x)$.

```
class MyFunc:
    def __init__(self, p0, p1, p2, ..., pn):
        self.p0 = p0
        self.p1 = p1
        ...
        self.pn = pn

    def f(self, x):
        return ...
```

Klasse for en funksjon med fire parametre

$$v(r;\beta,\mu_0,n,R) = \left(\frac{\beta}{2\mu_0}\right)^{\frac{1}{n}} \frac{n}{n+1}\left(R^{1+\frac{1}{n}} - r^{1+\frac{1}{n}}\right)$$

```
class VelocityProfile:
    def __init__(self, beta, mu0, n, R):
        self.beta, self.mu0, self.n, self.R = \
        beta, mu0, n, R

    def value(self, r):
        beta, mu0, n, R = \
        self.beta, self.mu0, self.n, self.R
        n = float(n)  # ensure float divisions
        v = (beta/(2.0*mu0))**(1/n)*(n/(n+1))*\
            (R**(1+1/n) - r**(1+1/n))
        return v

v = VelocityProfile(R=1, beta=0.06, mu0=0.02, n=0.1)
print(v.value(r=0.1))
```

Python-klasser: en grov skisse

```
class MyClass:
    def __init__(self, p1, p2):
        self.attr1 = p1
        self.attr2 = p2

    def method1(self, arg):
        # can init new attribute outside constructor:
        self.attr3 = arg
        return self.attr1 + self.attr2 + self.attr3

    def method2(self):
        print('Hello!')

m = MyClass(4, 10)
print(m.method1(-2))
m.method2()
```

Det er vanlig å ha en konstruktør som initialiserer attributter, men det er ikke et absolutt krav.

Python-klasser: en grov skisse

```
class MyClass:
    def __init__(self, p1, p2):
        self.attr1 = p1
        self.attr2 = p2

    def method1(self, arg):
        # can init new attribute outside constructor:
        self.attr3 = arg
        return self.attr1 + self.attr2 + self.attr3

    def method2(self):
        print('Hello!')

m = MyClass(4, 10)
print(m.method1(-2))
m.method2()
```

Det er vanlig å ha en konstruktør som initialiserer attributter, men det er ikke et absolutt krav.

Python-klasser: en grov skisse

```
class MyClass:
    def __init__(self, p1, p2):
        self.attr1 = p1
        self.attr2 = p2

    def method1(self, arg):
        # can init new attribute outside constructor:
        self.attr3 = arg
        return self.attr1 + self.attr2 + self.attr3

    def method2(self):
        print('Hello!')

m = MyClass(4, 10)
print(m.method1(-2))
m.method2()
```

Det er vanlig å ha en konstruktør som initialiserer attributter, men det er ikke et absolutt krav.

Eksempel

`id(obj)`: skriv ut entydig Python identifikator for et objekt

```
class SelfExplorer:
    """Klasse for å beregne a*x."""
    def __init__(self, a):
        self.a = a
        print('init: a=%g, id(self)=%d' % (self.a, id(self)))

    def value(self, x):
        print('value: a=%g, id(self)=%d' % (self.a, id(self)))
        return self.a*x
```

```
>>> s1 = SelfExplorer(1)
init: a=1, id(self)=38085696
>>> id(s1)
38085696

>>> s2 = SelfExplorer(2)
init: a=2, id(self)=38085192
>>> id(s2)
38085192

>>> s1.value(4)
value: a=1, id(self)=38085696
4
```

Eksempel: bankkonto

- Attributter: eiers navn, kontonummer, innestående beløp
- Metoder: innskudd, uttak, skrivUt

```
class Bankkonto:
    def __init__(self, navn, kontonummer, startbelop):
        self.navn = navn
        self.knr = kontonummer
        self.belop = startbelop

    def innskudd(self, belop):
        self.belop += belop

    def uttak(self, belop):
        self.belop -= belop

    def skrivut(self):
        s = '%s, %s, innestaende: %s' % \
            (self.navn, self.knr, self.belop)
        print s
```

Eksempel på bruk

```
>>> a1 = Bankkonto('Arne Arnesen', '19371554951', 20000)
>>> a2 = Bankkonto('Ole Olsen',  '19371564761', 20000)
>>> a1.innskudd(1000)
>>> a1.uttak(4000)
>>> a2.uttak(10500)
>>> a1.uttak(3500)
>>> a1.skrivut()
Arne Arnesen, 19371554951, innestaende: 13500
>>> a2.skrivut()
Ole Olsen, 19371564761, innestaende: 9500
```

Quiz 1

Hvilke av variablene nedenfor er (a) lokale variable; (b) klassevariable; og (c) globale variable?

```
a = 35.3
b = 6.32

class F:
    def __init__(self, a, b):
        self.a = a
        self.b = b
        self.c = a+b

    def f(self, x):
        res = self.a * x**2 + self.b * x + self.c
        return res

    def skrivut(self):
        s = 'a = %g, b = %g' % (self.a, self.b)
        print(s)

p = F(0,1)
print(p.f(0))
p.skrivut()
```

Quiz 2

Hva skrives ut her?

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

    def __call__(self, header):
        s = '%s: %s, %s' % (header, self.navn, self.adr)
        return(s)

p1 = Person('Ole Olsen', 'Inkognitogaten 18, Oslo')
p2 = Person('Hilde Aas', 'Trosterudveien 36, Oslo')
print(p1('Info'))
```

Quiz 2

Hva skrives ut her?

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

    def __call__(self, header):
        s = '%s: %s, %s' % (header, self.navn, self.adr)
        return(s)

p1 = Person('Ole Olsen', 'Inkognitogaten 18, Oslo')
p2 = Person('Hilde Aas', 'Trosterudveien 36, Oslo')
print(p1('Info')) # Info: Ole Olsen, Inkognitogaten 18, Oslo
```