Objektorientert programmering i Python

IN1000

Høst 2018 – uke 9

Siri Moe Jensen

Innhold uke 9
Mer komplekse strukturer

- Referanser versus objekter (repetisjon)

- "Dot-notasjon"
- Spesielle metoder i egendefinerte klasser
  - sammenligning
  - utskrift
- Samlinger av objekter i beholdere (containers) som liste, mengde og ordbok.

Seminartimen på gruppene
- Utvikling av kode for sentrale metoder i oblig 7
- ..

Resten av semesteret

- Semesterplan

- Obliger
  - oblig 7: Frist 29.10 – anbefalt levering 22.10
    - Kommenter "Klar for retting" før fristen!
  - Seminartimer
  - Fagutvalgets IN1000-seminar: 27.10
  - oblig 8: Frist 5. 11

Referanser og objekter, inkl Mentimeter spørsmål

Koden spørsmålene er hentet fra ligger i kodefiler på uke9-siden.

Spørsmålene var hva som ble skrevet ut ved hver `print()`.

Kjør den gjerne, evt i Pythontutor, for å se svarene!

Referanser til objekter

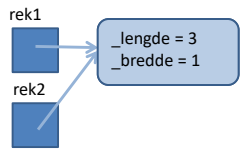
- Ofte trenger vi ikke tenke på at selve objektet ikke ligger i variabelen
- Men noen ganger må vi huske forskjellen på referansen og objektet
  - Hvis vi tilordner verdien fra en referansevariabel til en annen (objektet kopieres ikke – variablene refererer til samme objekt!)
  - om vi sammenligner to referanser

Sammenligning av referansevariable

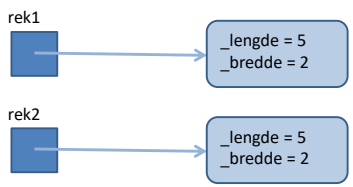
Kan sammenligne enten:

- Referansene, er det samme objekt?
  - Bruk `is` for å sjekke

```
rek1 is rek2
```



- Objektene, er det like objekter (likt innhold)?
  - Må sjekke instansvariablene i begge objekter!



Sammenligning av referansevariable

- Den som har skrevet klassen vet (bestemmer) hva som gjør to objekter "like".
- Vi kan lage en egen metode i klassen som gjør dette. Hvis metoden får navnet `__eq__` vil operatoren `==` sammenligne objekter av klassen slik vi bestemmer.
- Dette er gjort i Pythons List-klasse, slik at

```
liste1 == liste2
```

sammenligner alle elementene i en liste for oss

eksempel på __eq__ metode

```
class Rektangel :
  def __init__(self, len, bredde) :
    self._lengde = len
    self._bredde = bredde

  def __eq__(self, annen) :
    return (self._lengde == annen._lengde and
            self._bredde == annen._bredde)

r1 = Rektangel(8,6)
r2 = r1
print (r1 == r2)

r3 = Rektangel(6,4)
print (r3 == r2)
```

```
M:> rekt2.py
True
False
M:>
```

Spesielle metoder

- `__eq__` er en av mange "magiske" metoder som har en spesiell betydning i Python
- felles er
  - innledende og avsluttende dobbel underscore (__) i metodenavnet
  - kalles på andre måter enn ved metodenavnet
  - eks: `__eq__` når `==` brukes på objekter av klassen

Flere spesielle metoder

- Tabell 9.1 i boka viser en rekke andre spesielle metoder som kan implementere logiske (eks `==`, `!=`, `<`) og aritmetiske (eks `+`, `*`) operatorer for en klasse som trenger det
- `__init__` kalles ved opprettelse av nytt objekt, oppretter og initierer (gir startverdi til) instansvariablene
- `__str__` og `__repr__` gjør om objekter til strenger

Objekter som strenger

For å vise frem objekter (som strenger):

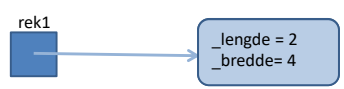
- `__str__`
  - kalles når vi bruker `print(s)` og `str(s)`
  - lager *brukervennlig* utskrift, lag den slik du ønsker
  - hvis den ikke finnes i klassen kalles `__repr__`

Lag gjerne denne

- `__repr__`
  - leverer en *komplett og entydig* representasjon av objektet
  - default: modul , klassenavn og minneadresse for objektet
  - kalles ved utskrift av elementene i en liste
  - eller når `__str__` ikke finnes i klassen

"Skrive ut" objekter

Printer vi en referanse, får vi en (ofte litt kryptisk) tekst ut:




```
print(rek1)
```

```
C:\Programmering>
C:\Programmering>python rektangel.py
<__main__.Rektangel object at 0x0000025FC328E6A0>

C:\Programmering>
```

resultat av `__repr__`

eksempel på __str__ metode

```
class Rektangel :
  def __init__(self, len, bredde) :
    self._lengde = len
    self._bredde = bredde

  def __str__(self) :
    penStreng = "Lengde: " + self._lengde + \
        ", bredde: " + self._bredde)
    return penStreng

r1 = Rektangel(8,6)
print (r1)
```

```
>python rek.py
Lengde: 8, bredde: 6
```

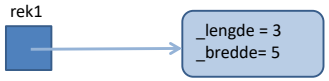
Oppsummert spesielle (magiske) metoder

- Ofte nyttig å skrive **`__str__`** og **`__eq__`** for egne klasser
- Hvis behov for å sortere f eks trengs flere logiske operatorer (<, >, != ,,,)

«dot-notasjon»

- For å endre (eller lese av) ***innholdet*** i ***objektet***, kaller vi på metoder for det objektet vi er interessert i. Metoden adresserer vi med «*dot-notasjon*» på referansevariabelen.

```
rek1.reduser(1,1)        # angir objekt og metode
print(rek1.areal())
```



«dot-notasjon» i flere ledd

- En referanse kan ligge i en variabel (som rek1) – eller være en returverdi fra en funksjon eller metode (som igjen kan være kallt på en referanse som var returverdi fra en metode – osv osv)

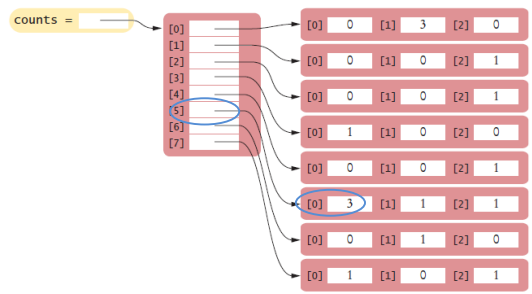
```
from rektangel import Rektangel
                                  # Oppretter nytt Rektangel objekt og
print(Rektangel(10,15).areal())   # kaller på metoden areal for dette,
                                  # printer til slutt returverdi fra areal
```

Samlinger av verdier (se uke 3)

- Beholdere (containers) er viktige verktøy i programmering
- Gjør det mulig å organisere og arbeide med samlinger av objekter
- Beholdere tilbyr ulike egenskaper – velges ut fra behov
- Så langt har vi sett på
  - Lister (List). Rekkefølge, nummerert
  - Mengder (Set). Unummerert, uten dubletter
  - Ordbøker (Dictionary). Par av nøkkel (typisk tekst) – verdi
- Objektene kan selv være samlinger – for eksempel lister

Liste av lister: Referanser i flere ledd

- `counts` inneholder en referanse til en liste
- listen inneholder referanser til andre lister
- innholdet i disse listene hentes ut med f eks counts[5][0]



Mange anvendelser for 2-dimensjonale strukturer!

Eksempler med fysisk 2-dimensjonal utstrekning

- sjakkbrett
- kart
- kommodeskuffer

eller mer abstrakte (skoleklasser 2A, 5C, 7B..)

Om innholdet i de innerste listene er lister i stedet for heltall, får vi en 3-dimensjonal liste, etc.

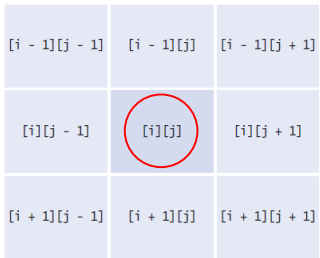
Kan tenke oss innholdet i en tabell

	0	1	2	3	4	5	..	..
0		[0, 1]						
1								
2								
3				[3, 3]				
4								
5								
..								
..								

nummerert som om vi leser en bokside

Liste av lister: Indeksering

- En tabell med generiske indekser:



Eksempel: Informatikk-emner (kurs)

- Vi skal lage et program for å velge informatikk-emner
- Initielle krav: Kunne liste opp alle emner med id (emnekode), antall poeng og høst eller vår-semester
- Designer en klasse Emne med instansvariable som over
- Bruker en liste for å organisere Emne-objekter

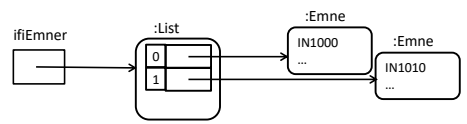
En klasse for emner

```
class Emne :
    def __init__ (self,emnekode,sem, stp) :
        self._emnekode = emnekode
        self._semester = sem
        self._studiepoeng = stp

    def __str__ (self) :
        linje = (self._emnekode + "(" +
                 self._semester + "): " +
                 str(self._studiepoeng) + " studiepoeng")
        return linje
```

Liste med objekter av egen klasse

- Eksempel: ifiEmner
- Hvert element i listen er (en referanse til) et emne-objekt

```
ifiEmner = []
ifiEmner.append(Emne("IN1000","host",10))
ifiEmner.append(Emne("IN1010","var",10))  # ..osv osv

for ettEmne in ifiEmner :
    print(ettEmne)              # kaller __str__()
```

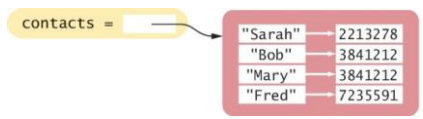



```
IN1000(host): 20 studiepoeng
IN1010(var): 20 studiepoeng
```

Opprette Dictionary (ordbok)

- Et program som slår opp telefonnummer
- Bruker en ordbok der navn er nøkkel, og telefonnummer er verdien

```
contacts = { "Fred": 7235591, "Mary": 3841212, "Bob":
   3841212, "Sarah": 2213278 }
```



Ordbok med referanser

- Verdiene i en ordbok (dictionary) kan være referanser til objekter
- Eksempel: Ordbok `ifiEmner` med emner
- Nøkkel (entydig): Emnekode
- Verdi: Referanse til et Emne-objekt

Ordbok med referanser II

```
ifiEmner = {}            #opprettet tom ordbok

ifiEmner["IN1000"] = Emne("IN1000","host",10)
ifiEmner["IN1010"] = Emne("IN1010","var",10)
# etc

for ettEmne in ifiEmner.values() :
    print(ettEmne)       # Kaller __str__()
```