IN1000
Høst 2024 – uke 12

Forberedelser til prøveeksamen og eksamen i IN1000

Siri Moe Jensen og Geir Kjetil Sandve

Dagens forelesning

- Undervisning i IN1000 frem mot eksamen
- Overordnet om pensum og læringsmål
- Gjennomgang av pensum høsten 2024
- Tips til arbeidet med prøveeksamen

IN1000 tilbud frem til eksamen (2.12)

- Undervisning mot eksamen
  - prøveeksamen tilgjengelig i dag -> 30.11
  - siste forelesning 13.11: Gjennomgang prøveeksamen + eksamenstips
  - gruppetimer (til og med 12.11)
  - repetisjonskurs (fra 14.11)
  - Fredags-python frem til eksamen

  - Detaljer om repetisjonskurs kommer på semestersiden (i dag)

- Andre ressurser
  - pensumbok
  - lysark og opptak fra forelesningene
  - innleveringer med tilbakemeldinger
  - eksamensoppgaver med løsningsforslag (NB: prøveeksamen ~fjorårets eksamen)

Hva skal vurderes? Fra emnesiden

Etter å ha tatt IN1000

- forstår du prinsippene for objektorientert programmering og kan benytte disse til å skrive enklere objektorienterte programmer

- kan du programmere i programmeringsspråket Python og kan bruke dette til å løse mindre problemer ved hjelp av valg, løkker, funksjoner, lister, klasser og objekter

- kan du skrive oversiktlige og lesbare programmer

- er du i stand til å sette deg inn i andres programmer, finne eventuelle feil i dem og modifisere dem

Pensum

NB: **Introduksjon i objektorientert programmering (Python er verktøyet)**

Overordnet pensum

- Kapittel 1-9 i Python for Everyone 2/e av Cay Horstmann og Rance Necaise (2. utgave, Wiley 2016, 3. utgave fungerer også)
- Innleveringer og det som er forelest (lysark fra forelesningene)

Forelesninger, prøveeksamen og obliger viser:

- hvilket stoff vi prioriterer fra pensum
- hvordan vi forventer at du benytter det.

På eksamen kan du få..

- Teorioppgaver
- Variasjoner av programmer du har sett før
- Oppgaver der du må kombinere stoff på måter du ikke har sett før

Sentrale begreper i objektorientert programmering

- Klasse
- Objekt
- Referanse
- Konstruktør
- Instansvariabel
- (Instans)metode
- Metodekall
- Grensesnitt (versus implementasjon)
- Innkapsling
- public
- non-public (privat)

- Gjennomgått i forelesning uke 7-9
- Brukt i eksempler i uke 10-12, samt i innleveringene 6-8.
- Tips: IN1000 engelsk-norsk ordliste

Kapittel 9 Objekter og klasser

9.1 Objektorientert programmering

Forelesning++: Uke 7
Oppsummering i pdf på uke7-siden

Samle data og metoder som behandler dem i samarbeidende *objekter*. Et objekt tilbyr et bestemt sett av tjenester i form av *metoder*. Disse metodene utgjør *grensesnittet* til objektet.

Hvilke tjenester et objekt tilbyr defineres av *klassen* til objektet.

Kapittel 9 Objekter og klasser

Forelesning++: Uke 7

9.2 Implementasjon

= Hvordan skriver vi klassen når vi har bestemt grensesnittet?

- Velge instansvariabler som representerer informasjonen hvert objekt skal huske og jobbe med. Instansvariablene defineres og initialiseres i konstruktøren.

- programmere innholdet i metodene, inkl eventuelle hjelpemetoder

Kapittel 9 Objekter og klasser

Forelesning++: Uke 7

9.3 Grensesnittet til en klasse

- Innkapsling: Objektene kan brukes når man kjenner deres offentlige grensesnitt, altså de metodene klassen tilbyr:
    - Hva gjør de, hvilke parametere trenger de, og hva returnerer de (om noe) NB: datatyper!
- Man trenger (bør) ikke kjenne til klassens implementasjon for å bruke objekter av klassen

Kapittel 9 Objekter og klasser

Forelesning++: Uke 7-12

9.4 Design av datarepresentasjonen

- Hva avgjør hvilke instansvariabler (datastruktur) vi trenger?
- Hvilke data skal være tilgjengelig (kunne hentes ut) i grensesnittet?
- Hvilke oppgaver skal objektene løse for oss som krever lagring av data av noe slag – *over tid,* dvs gjennom flere metodekall?
- Trenger vi tjenester/ data fra andre objekter, som vi må ha referanser til?

Kapittel 9 Objekter og klasser

Forelesning++: Uke 7+8

9.5 Konstruktøren

- Metoden __init__ kaller vi klassens konstruktør
- Konstruktøren kalles automatisk når vi oppretter et nytt objekt av klassen ved hjelp av klassenavnet – kalles aldri som en vanlig metode
- I konstruktøren definerer og initaliserer vi instansvariablene (datastrukturen) som hvert objekt får sin egen utgave av
- De må få en initialverdi – selv om vi noen ganger ikke vet før senere hvilken verdi de skal ha
- Konstruktøren har alltid en formell parameter self

Kapittel 9 Objekter og klasser

Forelesning++: Uke 7+8

9.5 Konstruktøren (forts.)

- Initialverdien til en instansvariabel kan bestemmes på en av flere tidspunkter:
  - Når vi skriver klassen:
    Samme verdi for alle nye objekter (f.eks. teller = 0, eller en tom liste)
  - Når vi oppretter nye objekter av klassen:
    Parameter til konstruktøren, bestemmes for hvert objekt (f.eks. navn på student)
  - Konstruktøren finner selv verdien (for eksempel ved å lese fra fil)
- Senere kan verdien til en instansvariabel i et objekt endres av andre metoder

Kapittel 9 Objekter og klasser

Forelesning++: Uke 8 +

9.6 Implementering av metoder

- Alle metoder må ha med self-parameteren (hvilket objekt er metoden kalt for/ skal jeg arbeide med denne gangen?)
- Alle instansvariabler aksesseres ved hjelp av parameteren self i metodene
- Metoder kan være offentlige (tilhøre grensesnittet) eller non-public ("hjelpemetoder", brukes bare av andre metoder i klassen – da starter navnet med _ som i instansvariabler)

Kapittel 9 Objekter og klasser

- 9.10 Objektreferanser (referanser)

Forelesning++: Uke 8

- Brukes for å "få tak i" et bestemt objekt

- Må i endel sammenhenger passe på forskjellen på referanse og objekt:
  - to referansevariabler kan referere til samme objekt (testes med r1 is r2)
  - innholdet i en referansevariabel endres som andre variabler (kan settes til å referere et annet objekt, r1 = r2)
  - innholdet (instansvariablene) i objekter endres med kall på metoder (r1.endre(5, 7))

- self, None. For å sjekke om en variabel ikke er None:

  if r1 is not None:

Litt mer om **self**

- En metode er definert i en klasse, og har tilgang til alle instansvariabler i objekter av klassen.
- Et program kan opprette mange objekter av samme klasse
- Hvert objekt har sitt sett av instansvariabler

- Det betyr at når metoden utføres, må den vite *hvilket objekt sine instansvariabler* den skal jobbe med!

- Det er dette **self** angir: *Hvilket objekt skal jeg jobbe med nå?*

- self-parameteren får verdien til referansen metoden kalles på (før .)

Kapittel 9 Objekter og klasser

9.11 Eksempel: En klasse for brøker

- Eksempelet er nyttig, men ikke pensum
- **Pensum: Seksjon 9.11.3** som handler om spesielle metoder
- Vi har brukt
  __init__
  __str__
  __eq__

Forelesning++: Uke 9

Kapittel 9: Objects and classes "Faktastoff"

Pensum: Kapittel 9.1 – 9.10, 9.11.3. 40 sider totalt

- 9.1-9.6
- 9.10
- 9.11.3:
- "alt" du trenger å vite om klasser og objekter i Python (ca 25 sider)

Oppsummeringer, faktabokser:

- Programming Tip 9.1 og 9.2
- Syntax 9.1 og 9.2
- Common error 9.1

- Hopp over Special Topics (9.1, 9.2 og 9.3)!

Kapittel 9: Objects and classes
Tips om fremgangsmåter, eksempler

9.7-9.9

- nyttig om testing og hvordan komme fra ide til ferdig objektorientert program

- How to 9.1, Worked example 9.2: Oppskrift og eksempler; Klassen Meny og klassen Bankkonto

Vi forventer ikke at du designer (finner frem til klassene og relasjonene mellom dem) et større system med flere klasser. Men du bør kunne forstå en slik struktur og kunne implementere (programmere) den.

BRUK av dette i ulike sammenhenger: Flere klasser, mange objekter

- Typisk stoff for en større oppgave på eksamen

Forelesning: Uke 9-11

- Nyttig å jobbe med flere eksempler, ulike strukturer
  - Oblig 7 og 8, prøveeksamen
  - T-bane og DNA-eksempler fra forelesning
  - Tidligere eksamener

- Merk forskjellen på å se en ferdig løsning, eller noen som livekoder – og det å jobbe selv med oppgaven.

- => ikke gi opp for raskt under trening, aksepter at du må gjennom flere runder med testing, feilretting, endringer og forbedringer. Det er dette du lærer av.

Hva betyr skoleeksamen i Inspera?

- Møt opp i Silurveien – se studentweb. 4 timer.
- all skriving og innlevering skjer i Inspera, på maskiner i Silurveien.
  Windows PC, norsk tastatur
- Lurt å huske passordet ditt

- Inspera gir noe syntaks-hjelp med innrykk og farger – ikke mer
- det er ikke tilgang til andre programmer eller internett (mao ingen kjøring av kode)

- ingen digitale hjelpemidler kan medbringes til plassen
- du kan ha med hva du vil på papir (vurder hva du har tid til å bruke)

- mat og drikke er ok

Inspera

- Se dokumentasjon/ info: https://www.uio.no/studier/eksamen/inspera/
- Les førstesiden ("Informasjon")
- Velg språk (øverst på siden)
- Du kan gå frem og tilbake i besvarelsen og endre svar så mange ganger du vil inntil levering
- Dersom det spørres om et heltall, skriv et heltall (ikke for eksempel 6.0). Generelt: Ikke legg på ekstra tegn i oppgave 1 og 2. Disse automatrettes.

Prøveeksamen - praktisk

- Tilgjengelig i Inspera nå
- Gir et inntrykk av eksamen – men ikke de samme oppgavene ☺
- Stort sett oppgaver fra fjorårets eksamen
- Noen ekstra oppgaver for å vise eksempler på flere oppgavetyper (merket "ny").

- Plan:
  - Åpent nå
  - Stenges lørdag 30.11
  - Gjennomgås (delvis) på forelesning neste uke

- **Bruk anledningen til å teste ut både det faglige OG arbeidsform (kun papir hjelpemidler)**
- **Gjerne sette av 4 timer og kjenne på tidsbruk**

Oppbygning prøveeksamen (og eksamen)

- Del 1 og 2: Automatrettede oppgaver.
  Tester kodelesing og –forståelse, teori, hvordan kode utføres

- Del 3: Prosedural programmering med problemløsing

- Del 4: Programmering av et objektorientert system med flere klasser og objekter som refererer til hverandre, teller typisk ca 50% av eksamen

- Del 5: En liten algoritmisk "nøtt"

Vurdering

- Poengene viser hvordan den enkelte oppgave vektes
- Bokstavkarakterer (A-F, der F er ikke bestått)
- Oppgaven, løsningsforslag og sensorveiledning legges ut på semestersiden senest ved sensurtidspunkt
- Hva vektlegger vi?
  - Vi krever ikke kommentarer på eksamen. Kan brukes om du ønsker å klargjøre noe i løsningsvalget
  - Opplagte slurvefeil ignoreres om de ikke er gjennomgående eller vanskeliggjør lesingen
  - Du trenger ikke teste på lovlig input om ikke oppgaven ber om det
  - Ikke bytt ut navn som er gitt i oppgaven

Typiske feil/ trekk/ problemer

- Logiske
  - feil i while/ if-tester (and/ or, < eller >=, operator-presedens)
  - returnerer på feil sted

- Bruker ikke metoder som allerede er skrevet der de kan løse et behov

- Ikke lest oppgaven godt nok (eks. print eller retur?)

- Tar "snarveier" i datastrukturen ("glemmer" å slå opp i ordbok/ liste)

Eksamensforberedelser

- Jobb på ditt nivå
  - Problemer med løkker, funksjoner/ parameter, enkle objektorienterte programmer? Sett av tid til hvert tema. Forståelse og trening
  - Tren på å løse oppgaver ferdig før du tester eller sjekker løsningsforslag. Hvorfor skjer dette og ikke dette?
  - Tenk over og test ut hva slags (papir-)hjelpemidler som fungerer for deg med begrenset tid

- Problemløsning
  - Selvstendig trening
  - Noen triks/ fremgangsmåter i boka
  - «hva ville jeg gjort uten datamaskin», «hvilke Python verktøy kan være nyttige her», beskriv fremgangsmåte i tekst/ pseudokode, tegne

Hvordan forberede seg til den store oppgaven?

Løs større oppgaver som trening på...:

- Lese tekst!! Få overblikk, tegn skisse - gå tilbake for å dobbeltsjekke og få med detaljer
- Kompleksitet. Skaff oversikt over datastrukturen og tren på veksling mellom detaljer og det store bildet.
- Test og tren på det grunnleggende. Vær nøye med å forstå hvordan ulike mekanismer virker, forstå hvorfor. Noen fremgangsmåter/ mønstre går igjen, men må tilpasses -> krever forståelse.
- Å finne fremgangsmåter for å løse en oppgave i en mer sammensatt struktur. Hvilke data har jeg tilgang til i datastrukturen (direkte eller via metoder)?

- For eksamenstrening: Papir.
- "Alt" fins i læreboka!
- Ikke bruk eksamenstid på å bla i store (ukjente) papirbunker

Tips til oppgaver med flere klasser

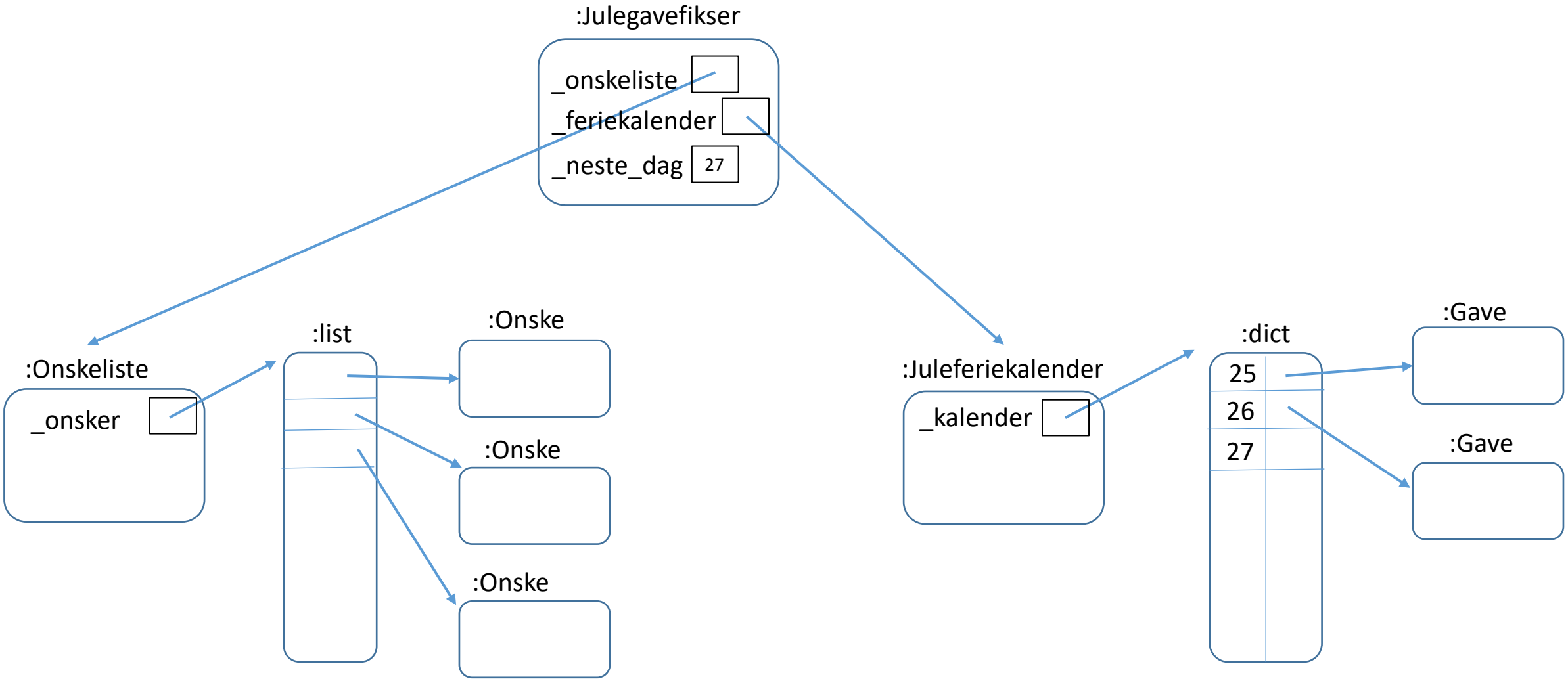
- Hva står eksplisitt i teksten om instansvariabler, metoder, parametere?
  => Trenger du noe mer?

- Tegn (skisse av) objekter og referanser for et tenkt eksempel under kjøring for oversikt

- Tenk "ett ledd av gangen". Tren på:
  - å bruke klasser som dere ikke har skrevet (ennå)
  - å skrive klasser som dere ikke vet hvor skal brukes (ennå)
  - begge deler krever at man forholder seg til et grensesnitt!

- Er noe uklart? Vurder alternativer, sjekk oppgaveteksten, skriv antakelser/ valg som kommentarer

"Hurtigskisse" datastruktur



Tips til oppgaver med flere klasser

Grensesnitt:

Sjekk mot oppgavetekst / koden din hva metoder du bruker fra andre klasser..

- tar som argumenter (verdier til parametere) og
- hva de leverer ut som returverdi:

Hva inneholder de, hvilken datatype?

Tips til oppgaver med flere klasser

- Velg variabelnavn som indikerer om
  - innholdet er en liste/ ordbok (flertall) eller en enkelt verdi
  - "enkle" verdier som string, int - eller referanse til et objekt

For eksempel

```
# personer: En ordbok med en samling personer
# personnavn: en string, kan f eks brukes som nøkkel i ordboken
# person: en referanse til person-objekt

for person in personer          # personer[personnavn] = person
    :
    :
```

Ordbok: Nøkler og verdier

```
ordbok = {"Januar": 31, "Februar": 28, "Mars": 31}
for antall_dager in ordbok.values():
    print(antall_dager)

for maned in ordbok:
    print(maned)
```

Hvordan kan du bruke både nøkkel og verdi for hvert enkelt element i en ordbok?

Ordbok: Nøkler og verdier

```
ordbok = {"Januar": 31, "Februar": 28, "Mars": 31}
for antall_dager in ordbok.values():
    print(antall_dager)

for maned in ordbok:
    print(maned)

for maned in ordbok:
    print(maned, "har",  ordbok[maned], "dager")
```

Hvordan kan du bruke både nøkkel og verdi for hvert enkelt element i en ordbok?

```
Januar har 31 dager
Februar har 28 dager
Mars har 31 dager
```

Hva er lettest å holde oversikt over?

```
for antall_dager in ordbok.values():
    print(antall_dager)
```

```
for maned in ordbok:
    print(maned)
```

```
for maned in ordbok:
    print(ordbok[maned])
```

```
for x in ordbok.values():
    print(x)
```

```
for x in ordbok:
    print(x)
```

```
for x in ordbok:
    print(ordbok[x])
```

Kunne vi gjort noen av navnene enda mer beskrivende?

Hva er resultatet av hver av disse linjene?

```
x = input("Svar: ").strip().split().lower()
```

```
x = input("Svar: ").lower().strip().split()
```

```
x = input("Svar: ").strip().lower().split()
```

```
x = input("Svar: ").split().lower().strip()
```

Hvordan kan vi gjøre det enklere for oss selv om vi mister oversikten?

Hva bidrar .strip() med i disse eksemplene?

Returnere eller skrive ut (printe)?

- Om oppgaven beskriver en metode: Står det at den skal returnere eller skrive ut på terminal (printe)?

- Om en metode (bare) printer ut en verdi er ikke verdien tilgjengelig for bruk på kallstedet (der metoden ble kalt)

- __str__ skal alltid *returnere* en streng (ikke skrive den ut)

Typisk bruk av while-løkker

- Ordreløkke der bruker skal kunne gi ordre til et program inntil det avsluttes

```
ordre = input("Ordre: ")
while ordre != "Avslutt":
    # utfoer ordre, gjerne kalle metode(r)
    ordre = input("Ordre: ")
```

  - pass på at du spør om ordre minst en gang
  - alltid endre inne i while-løkken det som testes (her: Be om og les inn ny ordre)

- For å teste input-verdi og be om ny dersom den ikke oppfyller visse krav. Da legger man gjerne til en feilmelding om hva som var galt før man ber om ny input.

```
navn = input("Oppgi navn: ")
while not navn.isalpha():
    print("Navnet kan kun inneholde bokstaver - prov igjen!")
    navn = input("Oppgi navn: ")
```

Typisk bruk av while-løkker

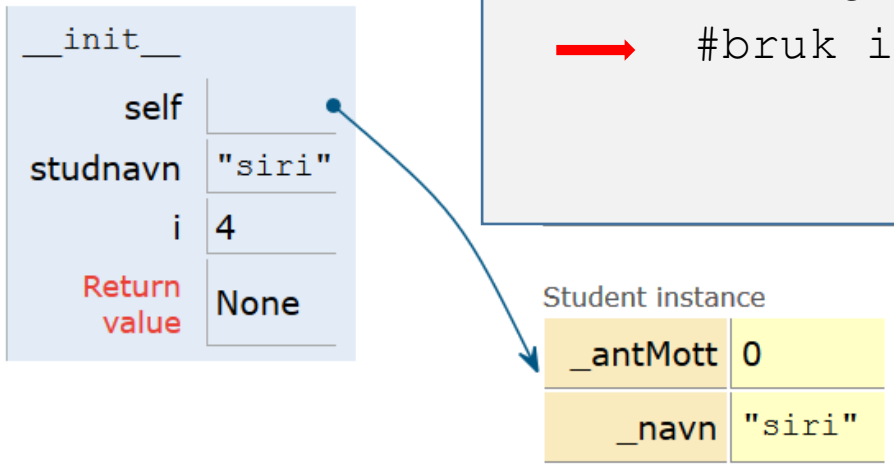
- En ordreløkke kan inneholde en test og evt feilmelding, for eksempel:

```
ordre = input("Ordre: ")
while ordre != "Avslutt":
    # Hvis lovlig ordre: utfoer, gjerne kalle metode(r)
    # ellers: Skriv en feilmelding
    ordre = input("Ordre: ")
```

=> Tenk over om koden vil kommunisere fornuftig med bruker både første gang og senere, både med korrekt og feil input.

Instansvariabler, lokale variabler, parametere

```
class Student:
  def __init__(self, studnavn):
    self._antMott = 0
    self._navn = studnavn
    i = len(studnavn)
    #bruk i til noe...
```



Instansvariabel, parameter, lokal variabel

- I en metode har vi tilgang til:
  - instansvariablene i et objekt (ved hjelp av .self).
  - Disse **opprettes** i konstruktøren (og bare der)
  - Når en metode avslutter, finnes fortsatt instansvariablene med sine verdier

  Dessuten (akkurat som i vanlige funksjoner og prosedyrer)
  - eventuelle parametere til metoden (eks: lengde, bredde). Disse oppstår når metoden blir kalt, og får verdi fra argumentene som sendes med. Når metoden er utført forsvinner de.
  - eventuelle lokale variabler metoden definerer (eks: mitt_areal). Som parametere, men må gis verdi inne i metoden.

Hva er instansvariabler, parametere og lokale variabler her?

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

    def areal(self):
        mitt_areal = self._lengde*self._bredde
        return mitt_areal
```