INF1001 Høst 2016

Uke 13:
Repetisjon og pensumgjennomgang

Innhold

1. time:
   Guidet tur gjennom læreboken (og pensum):
   Sentrale konsepter og mekanismer i Python

2. time:
   - Hva trenger vi utover å lese pensum?
     - Hva vil det si å kunne programmere?
     - Løsing og diskusjon av to konkrete eksempler på oppgave
   - Eksamenstips og informasjon om prøveeksamen

Hva skal evalueres? Fra kurssidene

Etter å ha tatt INF1001

- kan du skrive små til middels store programmer oppdelt i klasser
- har du grunnleggende ferdigheter i objektorientering i Python med klasser, metoder, objekter og referanser
- kan du lage programskisser med enkle UML klassediagrammer
- kan du lage løsninger på mindre, virkelige problemer på én maskin med brukerinteraksjon og lagring av data på disk
- kan du bruke enkle deler av Pythons standardbibliotek
- kan du finne og rette feil i egne programmer

Overordnet pensum

- Kapittel 1-9 i *Python for Everyone 2/e* av Cay Horstmann og Rance Necaise (2. utgave, Wiley 2016).
- *Personopplysningsloven*
- Obliger og det som er forelest (dvs lysark fra forelesningene)
- Anbefalt bakgrunnslitteratur: *Digitale medier* av Hannemyr et. al. (3. utgave, Universitetsforlaget 2015).

Forelesninger, prøveeksamen og obliger viser:

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

På eksamen kan du få:

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

Hvorfor har vi en lærebok?

- Læreboken forteller stort sett mer detaljert enn forelesningene.
- Det er en fordel å få ting fortalt på flere ulike måter.
- Læreboken er bedre å slå opp i.
- Random fact og Special topic er artig å lese og utdyper faget, men er ikke pensum i dette emnet
- Common error, Programming tip er nyttige og oftest pensum

Kapittel 1: Introduction

- Innholder først og fremst bakgrunnsinformasjon for det som kommer senere. Nyttig også i senere emner, og antakelig lettere å forstå nå enn da dere startet.
- «Programmering er problemløsing» [PFE: 1.7]
- Nyttig lærdom: Det første viktige steget i programmering er å omforme problemet til en algoritme, dvs gi det en form som datamaskinen kan løse. Så kan algoritmen skrives i Python

Kapittel 2: Programming with numbers and strings

- Variabler og tilordninger
- Numeriske uttrykk og typer
- Lite vekt på behandling av matematiske uttrykk i INF1001 – men viktig og nyttig for noen av dere senere.
- Strenger (tekst), konvertering og streng-metoder
- Innlesing fra tastatur (input)

Kapittel 3: Decisions

- Dette kapittelet tar for seg det som har med valg å gjøre:
  - if-setninger
  - ulike typer sammenligninger
  - boolske variabler, operatorer og uttrykk
- Dette må dere kunne bruke i programmering; unntaket er flytskjemaer [PFE: 3.5] som ikke er pensum.
- Merk at det er flere special topics i kapittelet som ikke er pensum (men som likevel kan være nyttige)

Kapittel 4: Loops

- Dette er uunnværlige redskaper i en programmerers verktøykasse. Omtrent alle programmer inneholder en løkke.
- while-løkker går så lenge en betingelse gjelder
- for-løkker går gjennom hvert element i en samling (f.eks. en liste)
  - Et spesialtilfelle er å gå gjennom en samling tall som kan brukes som indekser i en annen liste: `range(0,len(min_liste))`
- Det er mange "applications" og lignende som ikke er direkte pensum, men som kan være nyttig

Kapittel 5: Functions

- Nå begynner det å bli mer avansert. Alt i kapittelet er sentralt pensum (unntatt rekursjon [PFE: 5.10])
- [PFE: 5.1] om funksjoner som «svarte bokser» for lettere å holde oversikten over programmet.
- [PFE: 5.2] Hvordan lager og bruker vi funksjoner
- [PFE: 5.3] Parametre - noe av det aller viktigste i hele kurset!
- [BJ: 5.4–5.5] Returverdier (det som skiller funksjoner fra prosedyrer)
- [PFE: 5.7] Hvordan funksjoner bør være.

Kapittel 5: Functions *(forts)*

- [PFE: 5.7] Problemløsing med stegvis forfining beskriver en god teknikk til å la et program bli til litt etter litt.
- [PFE: 5.8] En variabels skop er delen av et program en variabel er tilgjengelig.
  - Enkelt sagt: En lokal metodevariabel finnes bare når metoden utføres.
  - Tenk også i forhold til OO

Kapittel 6: Lists

- Lister er en veldig mye brukt datastruktur.
- [PFE: 6.1] introdusere lister.
- [PFE: 6.2] viser ulike operasjoner på lister (hvor kun appending Elements er direkte pensum)
- [PFE: 6.3] gir eksempler på bruk av lister
- [PFE: 6.4-6.7] er ikke direkte pensum, men
  - poenget i ST3 (i 6.4) må man ha fått med seg (at f.eks. en liste sendt inn som parameter kan bli endret)
  - Kapittel 6.6 er veldig nyttig (også for eksamen) om hvordan tenke når man skal finne løsning til et problem.

Kapittel 7: Files and exceptions

- [PFE: 7.1-7.2] Hvordan lese fra og skrive til filer. Vi har i hovedsak basert oss på itererering av linjer (7.2.1)
- [PFE: 7.3-7.6] Ikke pensum. (kommandolinje-argumenter, binærfiler, unntakshåndtering)

Kapittel 8: mengder (set) og ordbøker (dict)

- [PFE: 8.1] Mengder er ikke direkte pensum, men selvsagt lov å bruke og kan være nyttig
- [PFE: 8.2] Ordbøker (dict) er pensum og nyttig i mange sammenhenger
- [PFE: 8.3] Mer komplekse strukturer, f.eks. ordbøker med lister som verdier. Også brukt sammen med OO (mer detaljer i senere slides).

Kunnskap
Fremgangsmåter, eksempler

Kapittel 9: Objects and classes

9.1-9.6 og 9.10:
- "alt" du trenger å vite om klasser og objekter i Python
- hopp over Special Topics 1,2 og 3

9.8-9.9
- nyttig om hvordan komme fra ide til ferdig oo program

9.11
- stort eksempel med en del stoff utenfor pensum, MEN 9.11.3 (Special Methods) er pensum

OO programmering: Fremgangsmåte

(mer detaljert i How-to.. 9.1)

1. Finn ut hvilke klasser vi trenger og hvordan de skal henge sammen; her er UML-diagram nyttig. **Husk**: En klasse skal representere noe spesifikt, enten et konsept eller en fysisk gjenstand.
2. Definer grensesnittet (dvs metodene) til klassen(e).
3. Finn representasjonen (dvs instansvariable) i klassen.
4. Programmer grensesnittmetoder og konstruktør for klassen
5. Lag og kjør et testprogram som tester klassen

UML klassediagrammer

- Bruk: Planlegging av programmeringen + dokumentasjon
- Viser: Klassene og deres relasjoner til hverandre
- Omfatter: Klassenavn og relasjoner, ofte også:
  - Grensesnitt (hvilke metoder kan kalles for et objekt av klassen)
  - Representasjon (hvordan er data representert i klassen)

Relasjonsangivelsen (navnet og antallet) gir et grunnlag for å definere mye av representasjonen.

Pensum: lysark 23-31, uke 9

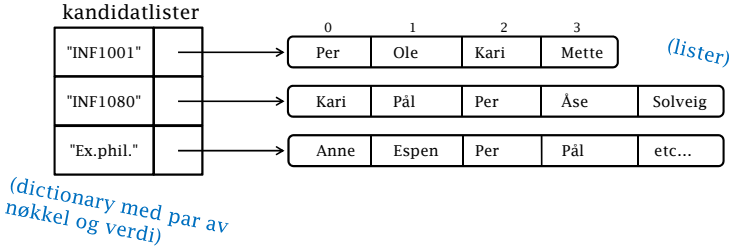
Hvordan organisere mengder av verdier?

- Plasserer verdiene i en samling (container)
- Verdiene som organiseres kan være f. eks. heltall eller objekter
- Liste: Enkel å bruke, holder orden på en rekkefølge og kan manipuleres gjennom indekser, ved gjennomløp eller første/ siste element. Kapittel 6
- Dictionary (=map, oppslag): Organiserer par av nøkkel og verdi, der hver nøkkel er unik og gir en verdi ved direkte oppslag. Ingen bestemt rekkefølge på elementene. Kapittel 8.2

Ikke pensum:
- Set: Kun unike verdier, ingen nummerering – effektiv prosessering. Kapittel 8.1

Samlinger av samlinger (kap 8.3.2)

- innholdet i en samling kan selv være samlinger
  - lister
  - dictionaries



Samlinger av samlinger

- Typisk stoff for "stor oppgave" på eksamen
- Nyttig å jobbe med flere eksempler
  - Uke 12 (fra eksamen 2014) med løsningsforslag
  - Uke 13 (eksamenssett 2015) gjennomgås seminartime
  - Uke 14 (prøveeksamen)
- Tegn for å beholde oversikt, tenk "ett ledd av gangen"

Digital representasjon

1. Tegnsett: Nyttig, men ikke viktig del av pensum

Det finnes mange tegnsett; de mest brukte er ISO Latin-1 (= ISO 8859-1) og Unicode; den siste kodes gjerne som UTF-8.

2. Tall/ tallsystemer

Dere bør kunne konvertere små tall mellom desimalt, binært og heksadesimalt

3. Digital koding av lyd og bilder er ikke pensum i INF1000

Se ressurssiden for uke 11:
- Dags lysark og screencast
- Notater med to ulike fremgansgmåter for konvertering av tall mellom tallsystemer (binært <-> desimalt <-> <-> heks)

IT og samfunn, personvern

Dette er viktig i pensum - se **lysark uke 8, 1.time**

- Berører Personopplysningsloven (Poppl) det jeg ønsker å gjøre? (se figur fra forelesning)
- Definisjoner, krav og unntak – se Poppl (lovteksten gjøres digitalt tilgjengelig på eksamen)

Se også boken Digitale medier (Gisle M fl). Dette er almenkunnskap som dere vil ha mye glede av utover i studiet – og ellers.

Hva vil det si å kunne programmere?

- Mer spesifikt:
  - Hva er det dere forventes å kunne etter INF1000?
- Eller sagt på annen måte:
  - Hva er det dere må kunne for å gjøre det godt på eksamen?

Programmering er en ferdighet!

- Målet er å kunne anvende programmering til å løse problemer
  - Å forstå de ulike begrepene (som if og while) er en forutsetning, men ikke tilstrekkelig
  - Å lese boka er ikke nok - man må også trene på å løse mange ulike problemer
  - Alle skriftlige læremidler kan tas med på eksamen - ferdigheten må man ha opparbeidet selv

Programmeringens natur

- Fra første time:
  - Software development happens in your head, not in an editor" (Andy Hunt)
  - "Programming is all about problem solving. It requires creativity, ingenuity, and invention"

Programmering er både inspirerende og frusterende

- "*.. combining rich, flexible human thought with the rigid constraints of a digital computer exposes the power and the deepest flaws of both*"

	Styrke	Svakhet
Menneske	Kreativitet og intuisjon	Manglende nøyaktighet og husk
Datamaskin	Presisjon og kompleksitet	Ingen forståelse av formål/intensjon

Eksempel på oppgave
(lett omskrevet fra eksamen INF1000, høst 2013)

- *Du skal nå skrive en funksjon som har tre parametre som tar imot flyttalls-verdier. Funksjonen skal finne den minste av de tre parameterverdiene og returnere denne. Hvis metoden heter minst, så skal f.eks. programsetningen*
  `v = minst(3, 1.3, 2.6)`
  føre til at variabelen v blir tilordnet verdien 1.3.
- Prøv selv å skrive et komplett svar på denne (5 min)!

Et mulig svar

```
def minst(a, b, c):
    svar=a
    if b < svar:
        svar = b

    if c < svar:
        svar = c

    return svar
```

Et annet mulig svar

```
•def minst(a, b, c):
•   svar=a
•   if a <= b and a <= c:
•       svar = a

•   if b <= a and b <= c:
•       svar = b

•   if c <= a and c <= b:
•       svar = c

•   return svar
```


Et tredje mulig svar

```
•def minst(a, b, c):
•   svar=a
•   if a <= b and a <= c:
•       svar = a
•   elif b <= c:
•       svar = b
•   else:
•       svar = c

•   return svar
```


En noe vanskeligere oppgave

- Vi har gitt en fil med navn "bokstaver.txt", som har én bokstav per linje. Skriv et program som leser gjennom denne filen og
  - a) teller antall A-er i filen
  - b) regner ut andelen av bokstavene i filen som er A (antall A dividert med antall bokstaver totalt).
- Prøv å skrive ned koden for dette på papir (5 minutt)

Et mulig svar

```
•antallA = 0
•antallBokstaver=0
•filNavn = "bokstaver.txt"

•for line in open(filNavn):
•   bokstav = line.strip()
•   antallBokstaver +=1
•   if (bokstav == "a"):
•       antallA += 1

•print( "Antall A: ", antallA )
•print( "Andel A: ", 1.0*antallA/antallBokstaver)
```

Hva som krevdes for å løse oppgaven

- Se behovet for en løkke (while) for å lese bokstaver
- Kunne lese fra fil
- Se behovet for å sjekke hva en bokstav er (if)
- Se behovet for en A-teller som begynner på 0
- Oppg b: se at man trenger en ekstra teller for alle bokstaver
- *(Man kunne også løst ved å lese inn i en liste. Det ville vært mer arbeid, men ville også løst problemet.)*

INF1000 Eksamensforberedelser og -tips

Høst 2016
Siri Moe Jensen

Før eksamen I

- Trening i større programmer med komplekse strukturer, overblikk/ helhet:
  - Eksamensoppgaver (ressurssider, seminartimer uke 12+13)
  - Oblig 7 (kan den forbedres?)
- Enkeltdeler av pensum
  - Forelesninger, slå opp/ utdyp fra lærebok
  - Løs oppgaver fra Trix og evt lærebok
  - Uløste obligoppg? Nyttige tilbakemeldinger på leverte?
- Prøveeksamen!

Før eksamen II

- Gjennomfør prøveksamen!
  - Planlegg hjelpemidler
  - Sett av 4 timer om mulig
  - Lag en realistisk eksamenssituasjon!
  - Revurder evt hjelpemidler til eksamen
- Les eksamensreglementet, sjekk grundig hvor/ når du skal møte (kurssidene og studentweb), beregn god tid
- Følg med på beskjeder på semestersidene, og les eller videresend UiO-mail

Prøveeksamen 21. november

- **Oppmøte fra kl 12:00, start 12:15**
  - **Gruppe 1: Chill**
  - **Gruppe 2: Sed**
- 12:15 – 16:00 gjennomføres prøveeksamen mest mulig likt som eksamen for å gi erfaring med eksamensformen og programvaren.
- Det blir mulig å jobbe videre på en kopi av besvarelsen etter at selve prøveeksamen avsluttes.
- En løsning gjennomgås på seminartimen 24.11

Om besvarelsene fra prøveeksamen

- Foreleserne vil ha tilgang til disse
- De er (selvsagt!) anonyme – akkurat som eksamensbesvarelsene
- Hensikten er å få mer erfaring med digitale oppgaver før eksamen – f eks om automatisk retting fungerer ok
- Vi har ingen ønsker om å vurdere deg ut fra prøveeksamen!
  - Den og andre forberedelser vil gi nyttig læring mot eksamen
  - .. så ikke mist motet om ikke alt går like lett allerede her!

På eksamen

Hensikten er å vise sensorene hvilke deler av pensum du behersker

- Du kan gå frem og tilbake i besvarelsen og endre svar så mange ganger du vil inntil levering
- Foreleser(e) vil gå rundt etter ca 0.5-1.5 time, forbered spørsmål om uklarheter
- Koden som skrives vurderes av sensorer = mennesker. Skal ikke kompileres eller kjøres av maskiner.
- Les oppgaven grundig! Hva ber den (ikke) om?

1. De aller fleste får liten tid

- Hovedprinsipp: Ikke kladd. Men tegn/ skisser gjerne ved siden av.
- Oppgavesettet gir totalt maksimalt 100 poeng. Vurder tidsbruk på enkeltoppgaver opp mot antall poeng de gir
- Svar på det det spørres om – kort og presist. Ikke gjenta oppgaven.
- Vi krever ikke kommentarer på eksamen. Kan brukes om du ønsker å klargjøre noe (men husk at sensor kjenner oppgaven ganske godt)

2. Sensor vil deg vel!

- Hensikten er å se hva du behersker av læringsmålene – ikke å pirke på språk eller skrivefeil
- Vi leter etter hva du kan – men du må vise oss det (og det må svare på det oppgaven spør om)
- Har du ikke tid til å programmere i detalj *kan* pseudokode/ kort beskrivelse være bedre enn ingenting – men den må vise noen relevante tanker om hvordan du ville gått frem. (dvs mer enn å gjenta oppgaven...)

Lykke til!!

(og bruk Piazza, mail og gruppetimer om det dukker opp spørsmål fremover!)