Innhold

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

Overordnet pensum

- Kapittel 1-9 i *Python for Everyone 2/e* av Cay Horstmann og Rance Necaise (2. utgave, Wiley 2016)
- 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:

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

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

- [PFE: 2.1] Variabler og tilordninger, numeriske uttrykk og typer
- [PFE: 2.2] Lite vekt på behandling av matematiske uttrykk i INF1001 – men viktig og nyttig for noen av dere senere
- [PFE: 2.4] Strenger, konvertering og streng-metoder
- [PFE: 2.5] Innlesing fra tastatur (input)
- [PFE: 2.6] Enkel grafikk - brukt av oss som eksempel

Kapittel 3: Decisions

- Dette kapittelet tar for seg det som har med valg å gjøre
  - [PFE: 3.1] if-setninger
  - [PFE: 3.2] Relasjonsoperatorer (<, >, == osv..)
  - [PFE: 3.3] Nøstede if-setninger
  - [PFE: 3.7] Boolske operatorer (and, or), uttrykk og variable
- Alle må kunne bruke if-setninger i programmering!
- [PFE: 3.5] Flytskjemaer - ikke direkte pensum i seg selv, men kan være nyttig for å presist forstå kodeflyten
- 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 - nesten alle programmer inneholder en løkke
- [PFE: 4.1] while-løkker går så lenge en betingelse gjelder
- [PFE: 4.3] Viktig å skjønne hvordan kode presist kjører!
  - *(bruk gjerne blyant og papir selv, eller PythonTutor)*
- [PFE: 4.6] 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))`
- [PFE: 4.7] Nøstede løkker *(vi så på det ifbm data fra fil)*
- [PFE: 4.3/5/8-11] 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 hvor en variabel er tilgjengelig.
  - Enkelt sagt: En lokal variabel i en funksjon er bare tilgjengelig når denne funksjonen 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 viktig del av pensum, men selvsagt lov å bruke og kan være nyttig
- [PFE: 8.2] Ordbøker (dict) er viktig del av 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).

Kapittel 9 Objekter og klasser

9.1 Objektorientert programmering

Samle data og metoder som behandler dem i samarbeidende *objekter*. Et objekt tilbyr et bestemt sett av tjenester i form av *metoder*.

Hvilke tjenester et objekt tilbyr kommer an på *klassen* til objektet.

Kapittel 9 Objekter og klasser

9.2 Implementasjon
= Hvordan skriver vi klassen!

- Velge *instansvariable* som representerer informasjonen hvert objekt skal ta vare på og jobbe med. Instansvariablene initialiseres i *konstruktøren.*
- programmere innholdet i metodene, inkl eventuelle hjelpemetoder

Kapittel 9 Objekter og klasser

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).
- Man trenger (bør) ikke kjenne til klassens *implementasjon* for å bruke objekter av klassen

Kapittel 9 Objekter og klasser

9.4 Design av datarepresentasjonen

Hva avgjør hvilke instansvariable (datastruktur) vi trenger?

- Hvilke data bør være tilgjengelig (kunne hentes ut) i grensesnittet?
- Hvilke oppgaver skal objektene løse for oss som krever tilgang til data av noe slag – over tid, dvs gjennom flere metodekall?
- Trenger vi tjenester fra andre objekter, som vi må ha referanser til?

Kapittel 9 Objekter og klasser

9.5 Konstruktøren

- Metoden __init__ kalles 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
- De defineres med self. etterfulgt av instansvariablenavn

Hvordan lage egne klasser?

```
class Student :

  def __init__(self) :
    self._antMott = 0

  def registrer(self) :
    self._antMott = self._antMott + 1

  def hentOppmote(self) :
    return self._antMott
```

konstruktør

Kapittel 9 Objekter og klasser

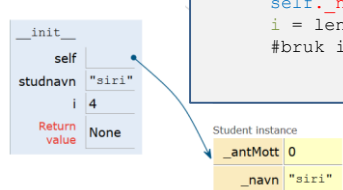
9.5 Konstruktøren (forts.)

- Initialverdien (startverdien) til instansvariable kan bestemmes på flere måter:
  - Når vi skriver klassen:
    Samme verdi for alle nye objekter (f.eks. teller = 0, 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 verdiene endres av andre metoder

Instansvariable, lokale variable, parametere

```
class Student :

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



Kapittel 9 Objekter og klasser

9.6 Implementering av metoder

- Alle metoder må ha med self-parameteren (hvilket objekt skal jeg arbeide med denne gangen?)
- Alle instansvariable aksesseres ved hjelp av parameteren self i metoden
- Kan være offentlige (tilhøre grensesnittet) eller private ("*hjelpemetoder*", brukes bare av andre metoder i klassen)

Kapittel 9 Objekter og klasser

9.7 Testing av klasser

- *Enhetstesting* (Unit testing)
- Sjekk at hver klasse, alle metoder, virker som planlagt før du bruker dem i større programmer. NB rekkefølgen på kall kan ha betydning fordi datastrukturen i objektene lagres (de har en *tilstand*)
- Kan gjøres i Python-tolken, men må da skrives inn på nytt for hver gang – bedre med testprogram
- Testen kan også skrives før du implementerer klassen (*testdrevet utvikling*, se uke 11)

Kapittel 9 Objekter og klasser

9.8 Problemløsing: Sporing av objekter

- Én metode for å kunne følge detaljert med på hva som faktisk skjer under kjøring av et program
- Andre metoder: Tavle med tegninger, plastelina, studenter, Pythontutor, enormt bra korttidsminne, ...
- Det viktige er å forstå og holde rede på hva som faktisk skjer med kontroll og data fra linje til linje.

Kapittel 9 Objekter og klasser

9.9 Problemløsing: Mønstere for objektdata

- Noen typiske behov for data går igjen: "patterns" eller mønstere.
- Kan raskere bruke tidligere erfaringer. Eks:
  - Å holde rede på en sum, eller en teller (utgifter)
  - Å samle flere verdier for eksempel i en liste (telefonkatalog)
  - Å holde rede på ulike egenskaper for et objekt (student)
  - Å holde rede på en posisjon (robot)
  - Å modellere tilstander for et objekt (hund, Rorbu)

Kapittel 9 Objekter og klasser

9.10 Objektreferanser

- Brukes for å holde rede på objekter
- Må vite når vi bruker referansen og når vi bruker objektet
- self, none
- Se uke 8, 9 og 10 inkl. illustrert, illustrerende eksempel (kommer på uke12 siden)

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__

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

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

9.11
- stort eksempel med en del stoff utenfor pensum,
  - NB: 9.11.3 (Special Methods) er pensum, og du bør kunne skrive og bruke metodene __str__ og __eq__ (se uke 9)

OO programmering: Fremgangsmåte

(mer detaljert i How-to.. 9.1)
(annet eksempel: Worked example 9.2)

1. Finn ut hvilke klasser vi trenger og hvordan de skal henge sammen, prøv deg frem med tegning og/ eller pseuokode. **Husk**: En klasse skal representere et abstrakt eller konkret fenomen
2. Definer grensesnittet (dvs metodene) til klassen(e).
3. Finn representasjonen (dvs instansvariable) i klassen. TEGN eksempler
4. Programmer grensesnittmetoder og konstruktør for klassen, evt også hjelpemetoder (private metoder)
5. Lag og kjør et testprogram som tester klassen

Flere klasser, mange objekter

- Typisk stoff for "stor oppgave" på eksamen
- Nyttig å jobbe med flere eksempler, ulike strukturer
  - Oblig 7 og 8
  - T-bane og DNA-eksempler fra forelesning uke 10+11
  - Student-eksempelet fra seminartimene uke 10+11
  - Prøveeksamen 2016 (seminartimer uke 12)
  - Prøveeksamen 2018 (forelesning uke 13)
- Tegn objekter og referanser for et tenkt eksempel for å beholde 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)

Hva vi gjennomgår i dag

- Første time: vandring gjennom læreboken
- Andre time: hva kreves utover å lese læreboken
  - Hva vil det si å kunne programmere?
  - Løsing og diskusjon av to konkrete eksempler på oppgaver
  - Gode råd for eksamen

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"

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.)*

IN1000 Eksamensforberedelser og -tips

Høst 2018
Siri Moe Jensen

Før eksamen I: Fag

- Fra problem til program + enkeltdeler av pensum
  - Løs oppgaver fra Trix og evt lærebok
  - Forelesninger, slå opp/ utdyp fra lærebok
  - Lag gjerne egne oppgaver / utvidelser/ variasjoner!
  - Uløste obligoppg? Nyttige tilbakemeldinger på leverte?
  - Praktisk trening OG teoretisk forståelse/ kunnskaper
- Trening i større programmer med komplekse strukturer, overblikk/ helhet:
  - Eksamensoppgaver
  - Oblig 7 og 8 (kan de utvides eller forbedres?)

Før eksamen II: Fag

- Tidligere eksamener: Nyttig og faglig relevant - MEN
  - Vektlegging kan være ulik årets eksamen
  - Noen oppgaver og løsningsforslag kan være preget av at de opprinnelig er gitt i Java
- Årets prøveeksamen: Legges så nært opp til årets eksamen som mulig i innhold og form – MEN
  - selvsagt forskjeller
  - pensum er fortsatt som beskrevet i dag!

Før eksamen II: Praktisk

- 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 6.-13. november

- Eksamenssettet åpnes etter forelesningen (14:15)
- Kan løses fra egen maskin eller Ifi-maskin, trenger bare bruker og nett (browser)
- Velger selv om du vil jobbe 4 timer i strekk, evt med tillatte hjelpemidler (som på eksamen) for å gi erfaring med tidsbruk og hjelpemidler i tillegg til eksamenssystemet.
- Prøveeksamen kan arbeides med i en uke (til 13.11 kl.12:00). Deretter vil du få rettet de delene som kan rettes automatisk (dvs flervalgsoppgaver, ikke programmer)
- Løsningsforslag blir gjennomgått på forelesning 13.11

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
- Programkoden som skrives vurderes av sensorer = mennesker. Skal ikke kompileres eller kjøres av maskiner.
- Les oppgaven grundig! Hva ber den (ikke) om?

De aller fleste får liten tid

- Hovedprinsipp: Ikke kladd - jobb i Inspera. 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 Fredags-Python om det dukker opp spørsmål fremover)