Uke 6: Overblikk og repetisjon

IN1000

Høst 2021

Siri Moe Jensen

Geir Kjetil Sandve

Innhold

Ingen nye Python nøkkelord denne uken!

- Planer for resten av semesteret
- Mentimeter-oppgaver fra gjennomgått stoff
- Hvordan bli en god programmerer -> hva er et godt program?

Semesteret i IN1000

- Uke 1-6: Grunnleggende elementer i programmering
- Uke 7-11: Objektorientert programmering, større programmer, mer kompleksitet
- Uke 12-14: Prøveeksamen, eksamenstips, repetisjon
- Uke 15: Eksamen (NB: Høsten 2021 vurderes IN1000 til Bestått/ Ikke bestått!)

Obligatoriske innleveringer

- Alt leveres i Devilry.ifi.uio.no (demo i gruppetime)
- Harde frister, stenges ved fristen
- Alle frister tirsdag kl 23:59
- Lever gjerne flere ganger, siste gjelder

- 1-6: ukes-obliger, disse gir poeng
  - I år ikke obligatoriske – men like nødvendige for å holde (og sjekke) progresjon!
  - Bruk denne uken til å komme ajour om det er noen du ikke har løst

- 7-8: større obliger, begge må godkjennes
  - Utsettelse ved egenmelding til gruppelærer/ retter
  - Nytt forsøk ved grensetilfeller for godkjenning (kort frist!)

- NB: Krav til selvstendig arbeid! Se info og oblig-reglement lenket fra obligsiden

Læringsmål for IN1000

- Fra [emnebeskrivelsen]

Hva lærer du?

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

Bestått/ Ikke bestått vurdering

- Hjemmeeksamen, 4 timer, i Inspera eksamenssystem

- Annen vurderingsform enn tidligere, MEN
  - $\Rightarrow$ Læringsmålene for emnet er ikke endret
  - $\Rightarrow$ Pensum er i praksis uendret (noe variasjon i hva som gjennomgås hvert år)
  - $\Rightarrow$ Tidligere eksamener gir relevant trening

- Kan bli noe endring i type og antall oppgaver (men ikke arbeidsmengde)
  - $\Rightarrow$ Eksamensforberedelser er tema de siste ukene
  - $\Rightarrow$ Prøveeksamen vil ligne årets eksamen

Tips for programmeringsstudier

- Programmering handler om å definere og løse problemer/ finne fremgangsmåter, dvs
  - Mye av læringen skjer mens du klør deg i hodet
  - Lær å like det ☺ - du jobber!

- Start koding selv om du ikke forstår alt, eksperimenter
- Men finn alltid ut hvorfor en feil "forsvant"!

- Jobb sammen med andre, diskuter løsninger og "hva skjer her"
- Jobb alene, svett over problemer ☺

- Du kan ikke uten videre hoppe over pensum i programmering

1

Learning that requires understanding

- Learning and teaching is NOT like plugging a pendrive into a learner's head.
- Learning is defined by BOTH the intended learning outcome of the teacher AND the learner's prior knowledge, experiences, and perhaps misconceptions.
- This is considerable work that the learner must do, though teachers may assist by various prompts.

on constructivism, adapted from UK-ICER20 keynote by Steve Draper, 3.9.20

Why and how is peer interaction so important in learning?

It does not rely on a peer telling you the right answer, nor on the soundness of their judgement.

It works by making the learner question themselves under the stimulus of a peer suggesting something different. A peer is often more effective at this than a teacher (or someone you judge ignorant): because with a peer, if you disagree, there is roughly a 50% chance of them being right or wrong, which forces you to think about issue rather than relying on the status of the person.

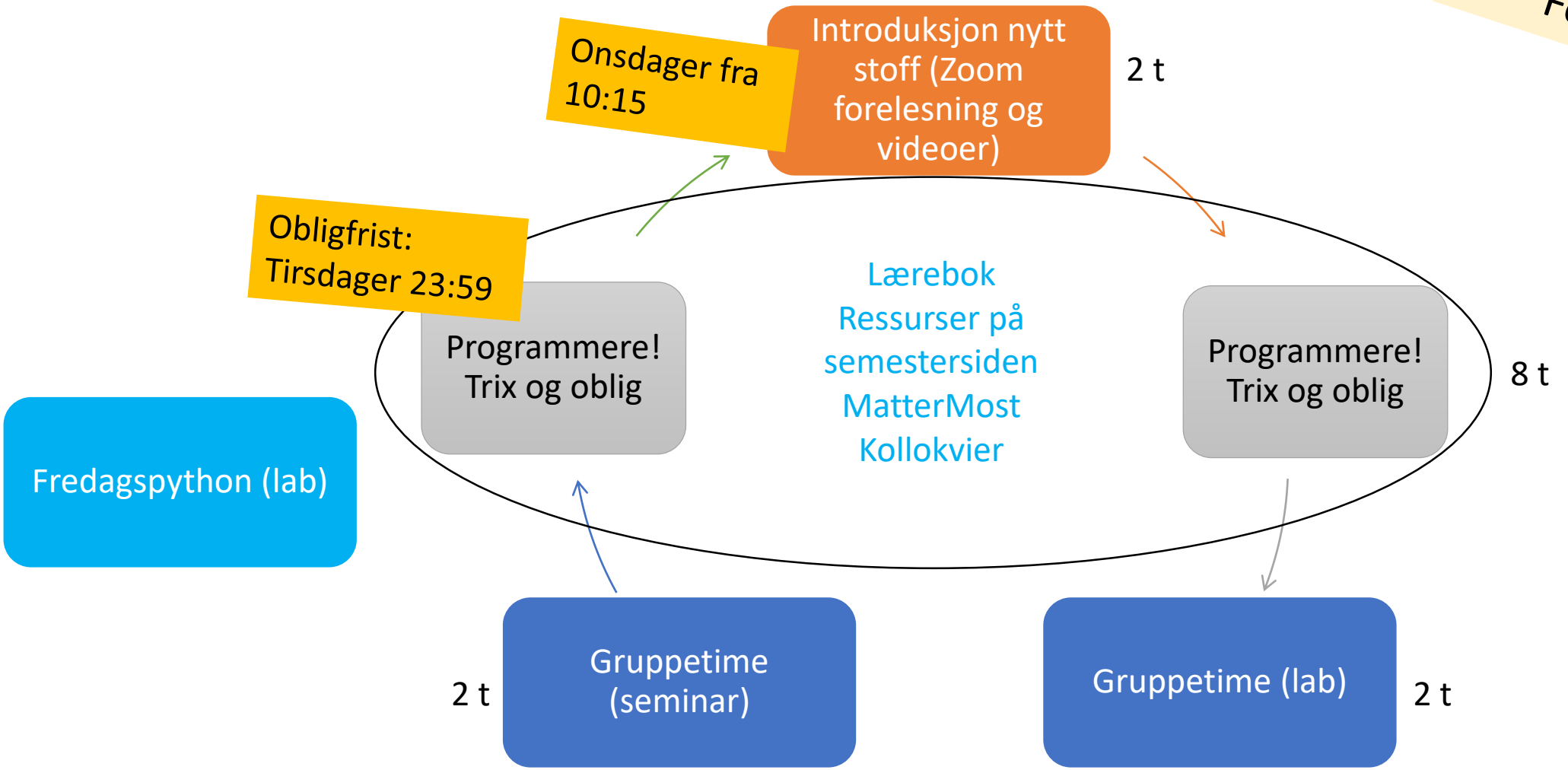
They act as a catalyst to your own thinking, not a source of pre-digested knowledge.

They prompt subsequent learning through a metacognitive mechanism.

UK-ICER20 keynote by Steve Draper, 3.9.20

Arbeidet i IN1000 – ca 13 t/ uke

Repetisjon:
- Innhenting
- Fordypning



Oppgaver fra uke 1-5

- "Code tracing" – å lese (egen) kode og "se" hva som skjer er helt nødvendig for å kunne skrive kode.

- Noen vanskeligheter fra obliger og eksamen

- Vi ønsker at dere skal bruke tid på å tenke over
  - hva som faktisk står i koden
  - hva som faktisk skjer på detaljnivå
  - hvilke feil det er lett å gjøre

- Bruker Mentimeter for å gi oppgaver som besvares individuelt og anonymt
  - gå til menti.com i et eget browser-vindu på PC eller mobil

Slik gjør vi det! Menti.com, kode:8807 3371

- For hver Mentimeter oppgave:
  1. Tenk gjennom: Hva mener du er rett? Hva kan gjøre at noen gir andre svar?
  2. Svar - ingen ser hva akkurat du svarer ☺
- Vi går gjennom hver oppgave underveis
- Fortsette med oppgaver etter pausen ved behov

Pause til 11:15

Menti.com

kode:8807 3371

Hva må vi kunne for å programmere?

- Problemløsning: Hvordan gå frem for å løse oppgaven?
  - Hva ville du gjort om du skulle løse den uten et program (men var like rask og flink til å utføre operasjoner som en datamaskin)?
  - Hvilke data må du holde rede på, er det noen sammenhenger mellom dem som må "huskes"?

⇒ Se gjerne *Problem solving* seksjoner og *How to* guides i læreboken

⇒ Trening!

- Hvordan uttrykke (oversette) en fremgangsmåte og representasjon til et programmeringsspråk?
  - Hvilke konstruksjoner/ språkmekanismer trenger du, hvilken datastruktur (variabler)?
  - Hvordan (helt nøyaktig) utføres disse når programmet kjører? Hva skjer med programflyt og variabler? => Semantikk
  - Hvilke tegn skal med i hvilken rekkefølge? => Syntaks

⇒ Forelesninger og gruppetimer, slå opp, trening!

Men jeg står fast!!

- prøv å tenke gjennom skrittvis, dele opp fremgangsmåte og data
- blir lettere med erfaring – dette ligner på noe jeg har sett før, løst før
- prøve og feile er helt normalt, viktig å starte et sted!

- løsningsforslag: Nyttigst **etter** at du har jobbet frem en egen løsning

Hva er riktig – og *god* – kode?

- Syntaks. Hvilke tegn skal med i hvilken rekkefølge i f eks en while-løkke?
- Semantikk. Når utføres innholdet i while-løkken? Hva skjer med programflyt og variabler?

- Logikk. Bruker vi den slik at den løser vårt problem?

Her virker programmet (er *riktig*)!!

- Pragmatikk, god design, Clean code, style ...: Hvordan bruke språket på en best mulig måte?
- Var en while-løkke beste valg – og er den skrevet slik at programmet er
  - lett å lese og forstå?
  - lett å endre?
  - (raskt å kjøre med effektiv bruk av prosessor og minne?)

Her kan programmet (nesten alltid) bli *bedre*

Style guide – PEP 8

- Det er skrevet en egen "style guide" for Python
- Gode tips om bl.a. navngiving og hvordan programmet bør "se ut" (layout):
  - navngiving
  - whitespace
  - kommentarer
  - linjelengde/ -deling

- Viktig: Dette er en guide
- Konsistens og bevisste valg er hovedpoenget – formålet er å skrive lettlest, lettforståelig, modifiserbar kode

Sterke meninger & pragmatisme - PEP 20

The Zen of Python

```
Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

God design - struktur

- Viktigere tema jo mer dere lærer
  - flere valg for samme problem
  - større programmer
  - mer samarbeid med andre
  - mer gjenbruk av kode

- Objektorientert programmering og design tilbyr verktøy som muliggjør god design i komplekse programmer med en bestemt måte å tenke på

Neste uke

- Hvorfor objektorientert programmering?
- Å designe, skrive og teste egne klasser
- Å bruke egne klasser i programmer

=> syntaks og overfladisk semantikk for objektorientert programmering