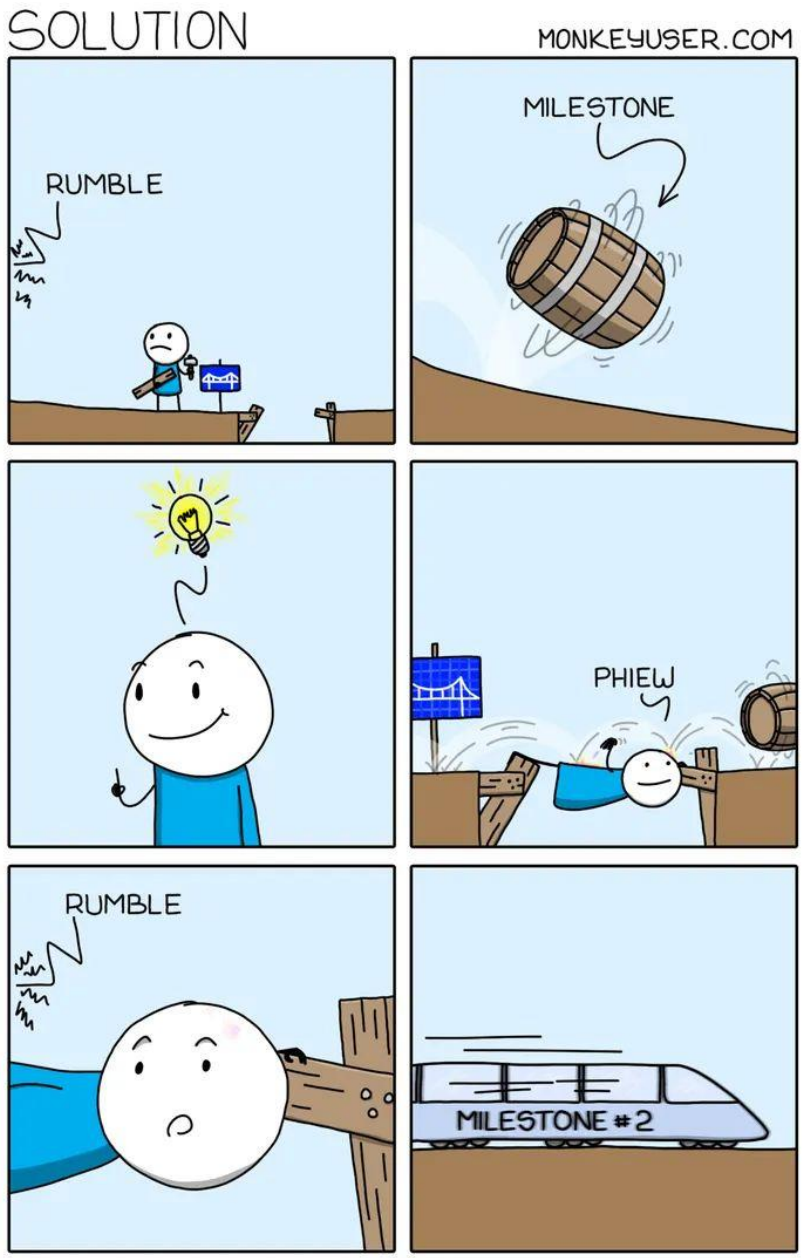
Uke 6: Overblikk og repetisjon

IN1000

Høst 2023

Siri Moe Jensen

Geir Kjetil Sandve



Dagens forelesning

Ingen nye Python nøkkelord denne uken!

- Parameteroverføring og variabelskop
- Mentimeter-oppgaver fra gjennomgått stoff

- Planer for resten av semesteret, eksamen

- Hvordan bli en god programmerer -> hva er et godt program?

- Midtveisevaluering

Uttrykk

(grundig gjennomgang i video på ukesiden)

Et utrykk er kode som evaluerer til en **verdi**
- som har en **type**

```
25
tall
[]
5+3
"Hei!"
legg_sammen(x, y)
False
sum > 5
False == True
input("Navn:" ).strip().split()
```

Prøv f.eks. **type(5)** og **type(5.0)** i python tolk eller Pythontutor

Der programmet trenger en verdi!

Du finner uttrykk f. eks. på høyresiden av tilordninger, eller som argumenter i et funksjonkall, eller inne i andre uttrykk

```
nytt_tall = tall

liste = []

sum = legg_sammen(3, 4)

if (sum > 5):
    ....
```

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

Menti.com

kode: 8245 0564

For hver Mentimeter oppgave:

Svar - ingen ser hva akkurat du svarer ☺

Når svarene vises - diskuter med naboen:

Hvorfor ble det slik?

Hva kan gjøre at noen gir andre svar?

Pythontutor

1: Opprette ny ordbok

2: Ordbok og logisk uttrykk

3: Nestet liste (ulike typer elementer)

4: Return fra løkke

5: Liste med lister

6: Legge sammen lister

7. Legge sammen lister i en liste

8: Evig løkke (uttrykk uten tilordning)

Time 2

Semesteret i IN1000 (se timeplanen)

- 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 (1. desember)

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

Bokstavkarakter A-F (F er ikke bestått)

- Digital skoleeksamen (Silurveien), 4 timer i Inspera eksamenssystem
  - Flervalgsoppgaver
  - Mest programmering
  - Enkel editor i Inspera for skriving av kode
  - Ikke mulighet for å kjøre koden, forståelse og problemløsningsevne vektlegges
- Eksamensforberedelser er tema de siste ukene
- Prøveeksamen i Inspera vil ligne årets eksamen og gir trening i både Inspera og å løse typiske eksamensoppgaver

Hvorfor er bare to innleveringer obligatoriske?

- Erfaringer høst 2022 (8 obligatoriske oppgaver, 1-6 med poengkrav):
  - Høy strykprosent
  - Endel var skuffet over resultatet

- Hvordan er det mulig å stryke på eksamen om man har løst obligene?
  - Tidspress
  - Ikke digitale hjelpemidler
  - Uvante oppgaver, uvant med eksamenssituasjonen
  - Ikke vant til å løse problemer selvstendig

- Lett å glemme at obligatoriske innleveringer ikke er for vår skyld
  - Det er prosessen (inkl bruk av tilbakemeldingene) som er produktet, ikke besvarelsen
  - Det kan være lett å overvurdere hvor selvstendig man har arbeidet, spesielt med mye hjelp fra medstudenter og gruppelærere

=> Ønsker å flytte oppmerksomheten fra godkjenning til læring

Unngå plagiatsaker
sikre læring!

Hva betyr selvstendig arbeid med innleveringer?

- Du har ikke kopiert hele eller deler av løsningen fra andre/ internett inkl AI
  - selvsagt ok å slå opp syntaks ("hvordan ser egentlig en for-løkke ut")
  - vær sikker på at du forstår hva koden gjør, og at du tilpasser den hvordan ditt program skal fungere.
  - ikke se på løsninger på den oppgaven du skal løse. Det begrenser hvilke fremgangsmåter du klarer å se for deg, og hindrer viktig trening i problemløsning

  Helt sikker: Bruk kun eksempler fra lærebok og forelesninger, legg dem vekk før du skriver egen kode

- Du har ikke samarbeidet for tett med andre
  - Vi anbefaler samarbeid om alt annet enn innleveringene
  - Om dere jobber sammen, ikke vis frem eller del ferdig Python-kode. Diskuter og del eventult løsninger på Trix-oppgaver

  Helt sikker: Arbeid alene med innleveringer

- Er du i tvil om du har kopiert / samarbeidet for mye: Oppgi hva + kilde i besvarelsen.
  - Kan gi godkjent eller nytt forsøk
  - Blir ikke regnet som plagiat

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

Hva må vi kunne for å programmere?

- Problemløsning: Hvordan gå frem for å løse oppgaven?
  - Forstå problemet/ oppgaven som skal løses!
  - 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

⇒ Erfaring!

- 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
- gå tilbake underveis og les oppgaven en gang til
- 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!

  => må regne med blindgater og flere runder

- løsningsforslag: Nyttigst **etter** at du har jobbet frem (i hvert fall med) en egen løsning

Hva er *korrekt* 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?

Veldefinerte, absolutte regler i språket

- Logikk.
  Bruker vi den slik at den løser vårt problem (...i alle aktuelle situasjoner)?

Krever presise krav og tester

Her virker programmet (er *riktig*)!

Hva er *god* kode (i tillegg til å være korrekt)?

- Lett å lese – oversiktlig – layout og navn gir god intuisjon.
  PEP 8 style guide dekker det meste. Viktig: Konsistens

Etablerte anbefalinger

- God design, clean code...:
  Hvordan bruke språket på en best mulig måte?

Ulike tilnærminger og "skoler" – men enighet om mye

  - Var en while-løkke beste valg – 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?)

Et program kan alltid (?) bli *bedre.* Nivå-avhengig.

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: PEP 8 er en guide

- Konsistens og bevisste valg er hovedpoenget – formålet er å skrive lettlest, lettforståelig, modifiserbar kode
- Mer bevisst egne valg når du jobber alene
- Viktigere jo større prosjekter du jobber i

Navngiving

- Se craftmanship-modul uke 4

Variabler og funksjoner:

- I læreboka (og Java): Påfølgende ord har stor forbokstav (mixedCase)
- PEP 8 (anbefalt Python): lowercase_with_underscores (snake_case)

Klassenavn (neste uke): Stor første bokstav i hvert ord (CamelCase)

- Er dette gode variabelnavn?

  I (stor 'i') l (liten 'L') O (stor 'o')

PEP 20 – et mer poetisk uttrykk ☺

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

- Prosedyrer og funksjoner er ett verktøy for bedre struktur

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

Midtveis-evaluering IN1000
(åpent spørsmål til slutt)

Menti.com

kode: 8245 0564

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