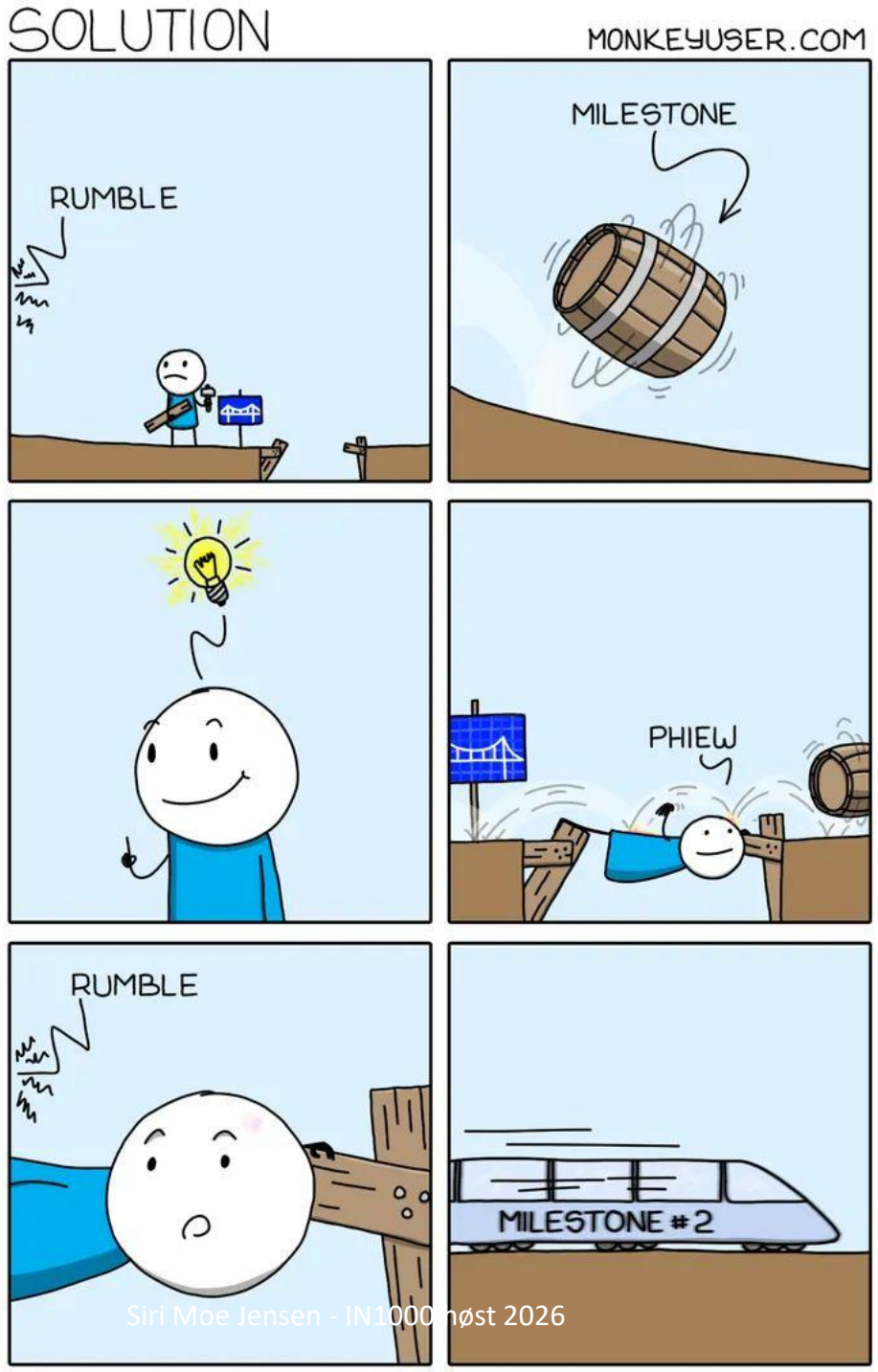
Uke 6: Overblikk og repetisjon

IN1000

Høst 2026

Siri Moe Jensen

Geir Kjetil Sandve



Dagens forelesning

Ingen nye Python nøkkelord denne uken!

- ✓ Parameteroverføring og variabelskop
- ✓ Midtveisevaluering av IN1000

- Resten av semesteret, arbeid med faget & eksamen
- Menti: Oppgaver fra uke 1-5
- 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 12-15: Prøveeksamen, eksamenstips, repetisjon og trening

- Uke 16: Eksamen (5.12)

Terskler i IN1000

1. Programflyt + språk + verktøy
2. Learning Edge Momentum
3. Objektorientert programmering
4. Veien fra forelesning/ oblig til eksamen

~Halvveis i pensum

- En uke til på oblig 5, mer omfattende med obligatoriske programmeringsoppgaver
- God kontroll på denne og tidligere innleveringer gir utbytte videre

De aller fleste vil møte nytt stoff fra neste uke:

- Gruppetimer, innleveringer og Trix
- Fritt oppmøte på alle grupper ut semesteret
  - Tilpass nivå
  - Gå på flere ved behov

Læringsmål for IN1000

Forelesninger og obliger viser hva vi prioriterer (kjernepensum)

Python for Everyone (kapittel 1-9) gir flere forklaringer, eksempler, oppgaver og illustrasjoner

- 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

Eksamen 4.12

- Digital skoleeksamen (Silurveien), 4 timer i Inspera eksamenssystem
  - Noen automatrettede oppgaver
  - Mest programmeringsoppgaver, kan være noe teori, forstå begreper (IN1000 ordliste)
  - Enkel editor i Inspera for skriving av kode

  => Ikke mulig å kjøre programmer, forståelse og problemløsningsevne vektlegges

- Hjelpemidler: Kun papir, ikke internett

- Karakter A-F (F er ikke bestått)

- Eksamensforberedelser de siste ukene

- Prøveeksamen i Inspera vil ligne årets eksamen og gir trening i både Inspera og å løse typiske eksamensoppgaver => åpnes i eksamenssystemet Inspera i begynnelsen av november

Hvordan er IN1000-uken lagt opp?

(se semestersiden/ Undervisningstilbud)

- Forelesning onsdag
- 1. gruppetime: Nye konsepter; syntaks, semantikk, bruk
- 2. gruppetime: Problemløsning
- Fredagspython: Lab med gruppelærer(e), sosial avslutning på uken

- Obligatoriske oppgaver:
  Påminnelse og viktig trening, teori (quiz) og praksis (programmering). Les og bruk tilbakemeldingene!
- Trix-oppgaver: Mengdetrening, på det du trenger å trene på.

- Pragmatisk programmering onsdag:
  Alternativ gjennomgang av ukens eller tidligere pensum

- Fordypningstilbud fredag:
  Litt mer krevende oppgaver

Hvorfor er gruppetimene viktige?

- 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

.. og hvorfor diskutere og jobbe med medstudenter?

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

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

Dessuten: å forklare, formulere og argumentere for en fremgangsmåte eller løsning gir klarere tenking, dypere forståelse og trening i bruk av fagterminologi

UK-ICER20 keynote by Steve Draper, 3.9.20

Sikre læring
Unngå fusk & plagiat

Hva betyr selvstendig arbeid med innleveringer?

- Du har ikke kopiert hele eller deler av løsningen fra andre/ internett / KI
  - selvsagt ok å slå opp syntaks og semantikk (hvordan virker/ skriver jeg en for-løkke)
  - forstå og tilpass koden til ditt program
  - løsningsforslag begrenser hvilke fremgangsmåter du klarer å se for deg, og hindrer 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 – men ikke *alltid*
  - Om dere jobber sammen, ikke del ferdig Python-kode. Diskuter og del eventuelt løsninger på Trix-oppgavene

  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 fusk

Formelt om fusk ved UiO

Hvorfor er mange oppgaver frivillige?

- De er like viktige/ nødvendige for å lære pensum
- Alt pensum bygger på foregående uker

- Verdien ligger i *arbeidet* med løsningen: Programmering
- Om du ikke har lagt arbeid i det du leverer (og tilbakemeldinger) er det
  - Liten verdi for deg
  - Sløsing med ressurser å rette

- *Frivillig* er en påminnelse om at
  - Ditt utbytte – er ditt ansvar

Repetisjon

Basics er ikke alltid basics

Verdier, uttrykk og variabler

- “a **value** is the representation of some entity that can be manipulated by a program”
- En **verdi** endrer seg ikke
- En verdi har (er av) en **type**
  - Prøv f.eks. **type(5)** og **type(5.0)** i python tolken eller Pythontutor
- Eksempler
  - 5 (int)
  - False (boolean)
  - 3.14 (float)
- Hva har disse felles?
  - 25
  - False
  - 3.0+0.14
  - input ("Hva heter du?")
  - False == True
- Et **uttrykk** er programkode som evaluerer til en verdi
- Uttrykk kan være sammensatte
  - int(input("Hvor gammel er du?"))

Verdier, uttrykk og variabler

- En **variabel** i Python har et navn og en verdi
- Variabler **får en (ny) verdi** når de står på venstresiden av en tilordning
- Eksempler
  - tall = 5
  - ferdig = False
  - pi = 3.14
- Når de brukes andre steder i programmet er variabler **uttrykk** (som evaluerer til variabelens verdi)
  - print (sum)
  - antall = antall+1

Hvor mange uttrykk har vi her?

Funksjonskall er uttrykk

Funksjoner kommuniserer med kallstedet gjennom parametere og returverdier

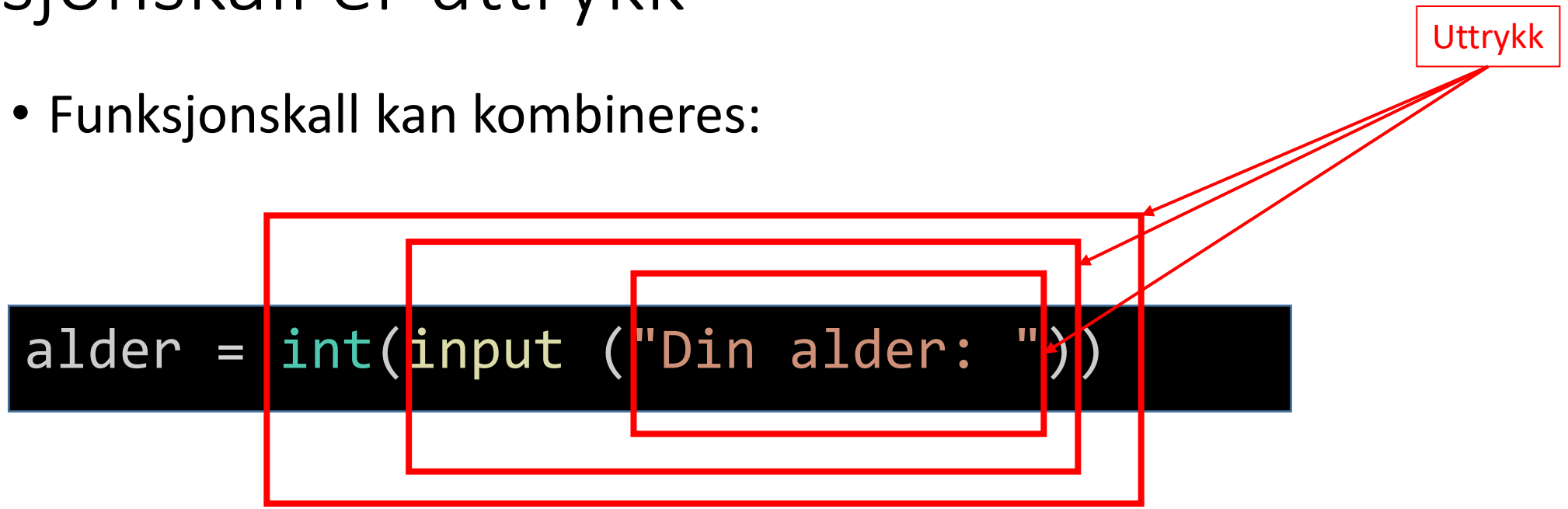
- Prosedyre tar 0-n argumenter – returnerer ingen verdi
- Funksjoner tar 0-n argumenter – returnerer én verdi

- Når vi trenger en verdi i et program har vi flere muligheter:
  - er det alltid den samme? Literal/ konstant
  - Har vi lagret verdien i en variabel?
  - Kan vi regne den ut?
  - Kan vi finne den ved å kalle på en funksjon?

Uttrykk

Funksjonskall er uttrykk

- Funksjonskall kan kombineres:



- funksjonen `input` returnerer en verdi som brukes direkte som argument til funksjonen `int`.
- `int` returnerer en verdi som legges i variabelen `alder`.

Repetisjonsoppgaver

Gjennomgås i sakte tempo i video på uke 6-siden

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

⇒ 2. dobbelttime på gruppene

⇒ Erfaring (=mengdetrening)!

- 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 (1. dobbelttime), slå opp, følg kode i Pythontutor, mengdetrening!

Men jeg står fast!!

- prøv å tenke gjennom skrittvis, dele opp fremgangsmåte og data
- gå tilbake underveis og sjekk hva oppgaven ber om – og oppgir

- 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

  => oppdag feil tidlig, test små biter av koden før du går videre

- Løs (flere) enklere oppgaver
- løsningsforslag: Nyttigst **etter** at du har jobbet frem en egen løsning

Hva er *korrekt* kode?

- Syntaks.
- Semantikk.

Veldefinerte, absolutte regler i språket

- Logikk.
  Bruker vi språket slik at vi løser problemet (...i alle aktuelle situasjoner)?

Krever presise krav og tester

Her virker programmet (er *riktig*)!

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

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

Ulike tilnærminger og "skoler" – men enighet om mye

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

Etablerte anbefalinger

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

God design

- ✓ Gir oversikt og lesbarhet
  - ✓ Mer effektiv utvikling, færre feil
- ✓ Muliggjør gjenbruk av kode
- ✓ Forenkler vedlikehold, utvidelser

- 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 tankesett og verktøy som muliggjør god design i komplekse programmer

Contents

- Introduction
- A Foolish Consistency is the Hobgoblin of Little Minds
- Code Lay-out
  - Indentation
  - Tabs or Spaces?
  - Maximum Line Length
  - Should a Line Break Before or After a Binary Operator?
  - Blank Lines
  - Source File Encoding
  - Imports
  - Module Level Dunder Names
- String Quotes
- Whitespace in Expressions and Statements
  - Pet Peeves
  - Other Recommendations
- When to Use Trailing Commas
- Comments
  - Block Comments
  - Inline Comments
  - Documentation Strings
- Naming Conventions
  - Overriding Principle
  - Descriptive: Naming Styles
  - Prescriptive: Naming Conventions
    - Names to Avoid
    - ASCII Compatibility
    - Package and Module Names
    - Class Names
    - Type Variable Names
    - Exception Names
    - Global Variable Names
    - Function and Variable Names
    - Function and Method Arguments
    - Method Names and Instance Variables
    - Constants
    - Designing for Inheritance
  - Public and Internal Interfaces
- Programming Recommendations
  - Function Annotations
  - Variable Annotations
- References
- Copyright

Page Source (GitHub)

PEP 8 – Style Guide for Python Code

Author: Guido van Rossum <guido at python.org>, Barry Warsaw <barry at python.org>, Alyssa Coghlan <ncoghlan at gmail.com>

Status: Active
Type: Process
Created: 05-Jul-2001
Post-History: 05-Jul-2001, 01-Aug-2013

▸ **Table of Contents**

Introduction

This document gives coding conventions for the Python code comprising the standard library in the main Python distribution. Please see the companion informational PEP describing style guidelines for the C code in the C implementation of Python.

This document and PEP 257 (Docstring Conventions) were adapted from Guido's original Python Style Guide essay, with some additions from Barry's style guide [2].

This style guide evolves over time as additional conventions are identified and past conventions are rendered obsolete by changes in the language itself.

Many projects have their own coding style guidelines. In the event of any conflicts, such project-specific guides take precedence for that project.

A Foolish Consistency is the Hobgoblin of Little Minds

One of Guido's key insights is that code is read much more often than it is written. The guidelines provided here are intended to improve the readability of code and make it consistent across the wide spectrum of Python code. As PEP 20 says, "Readability counts".

A style guide is about consistency. Consistency with this style guide is important. Consistency within a project is more important. Consistency within one module or function is the most important.

However, know when to be inconsistent – sometimes style guide recommendations just aren't applicable. When in doubt, use your best judgment. Look at other examples and decide what looks best. And don't hesitate to ask!

In particular: do not break backwards compatibility just to comply with this PEP!

Some other good reasons to ignore a particular guideline:

1. When applying the guideline would make the code less readable, even for someone who is used to

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

Neste uke

Husk å avslutte midtveisevalueringen om du ikke alt har gjort det!

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

=> syntaks og overordnet semantikk for objektorientert programmering