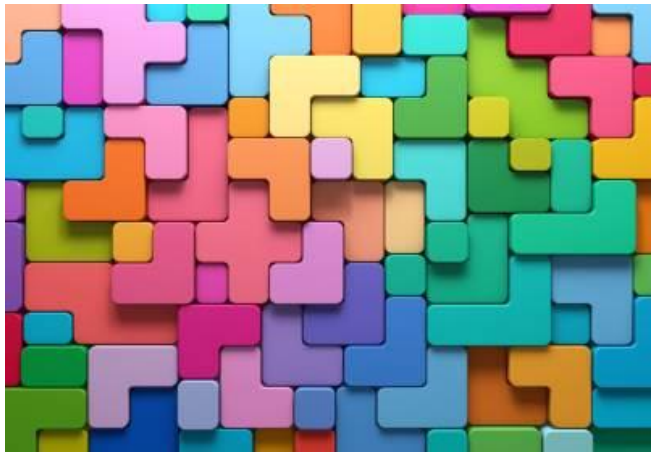
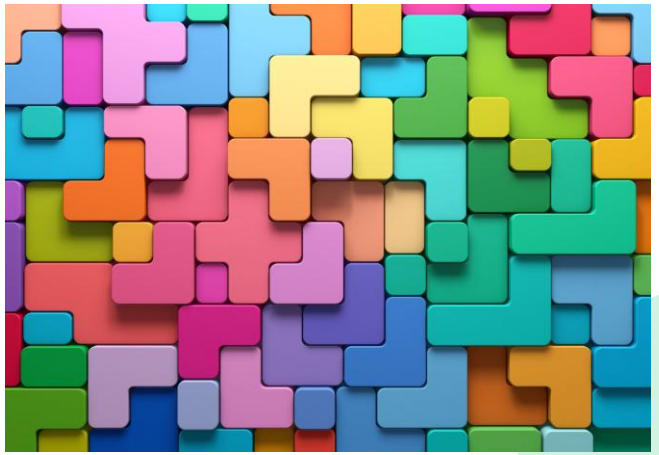
EXIT-KODER, LOGGING OG DEBUGGING

FORELESNING 5

MANDAG 1/9

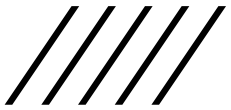


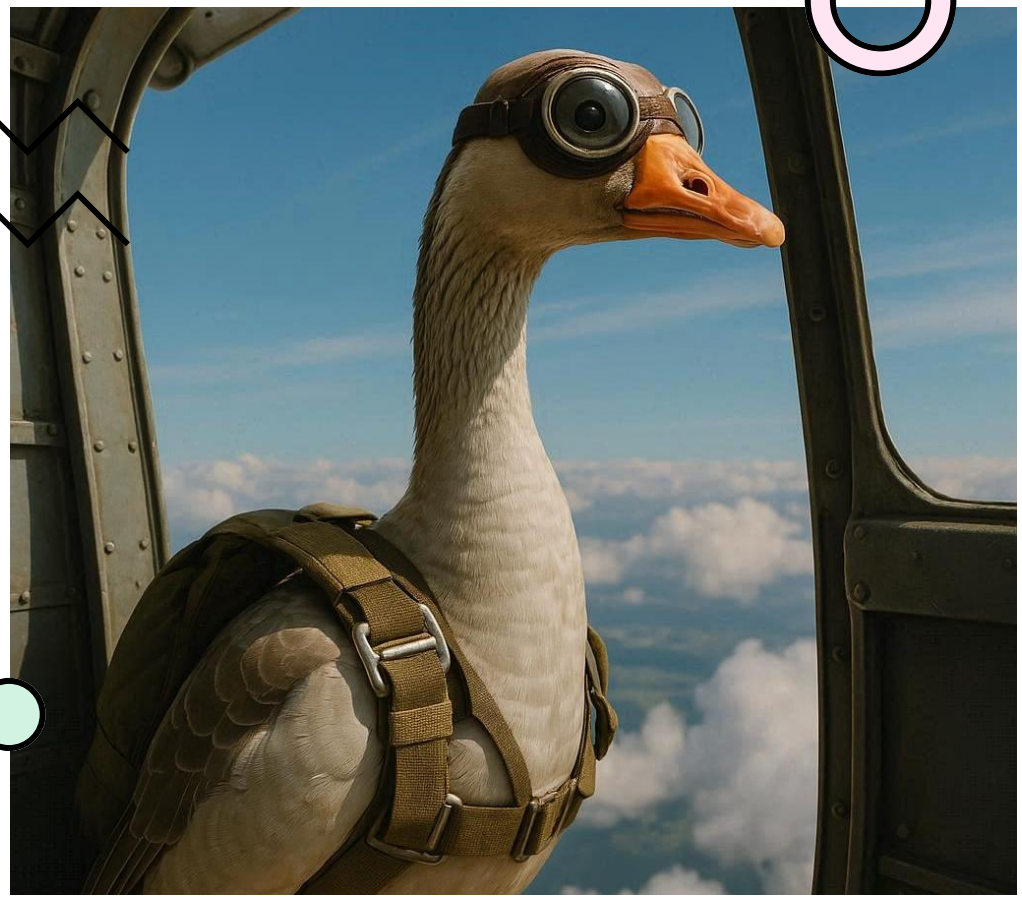


(bilder generert av bing image creator)

Exit-koder

- Noen ganger kjører et program et annet program
- Da kan det første programmet trenge å vite om det andre programmet feilet eller ikke
- Hvordan kan vi se hvordan det gikk med det forrige programmet som kjørte?
- Mac / Linux: **echo $?**
- Windows (PowerShell): **$LASTEXITCODE**
- **0** betyr "suksess", **1** betyr "noe gikk galt",
  2+ er forhåndsdefinerte feilkoder som man kan slå opp et sted





LIVEKODING: EXIT-KODER

HISTORIC ARTICLE

Sep 9, 1947 CE: World's First Computer Bug

On September 9, 1947, a team of computer scientists reported the world's first computer bug—a moth trapped in their computer at Harvard University.

GRADES
3 - 12

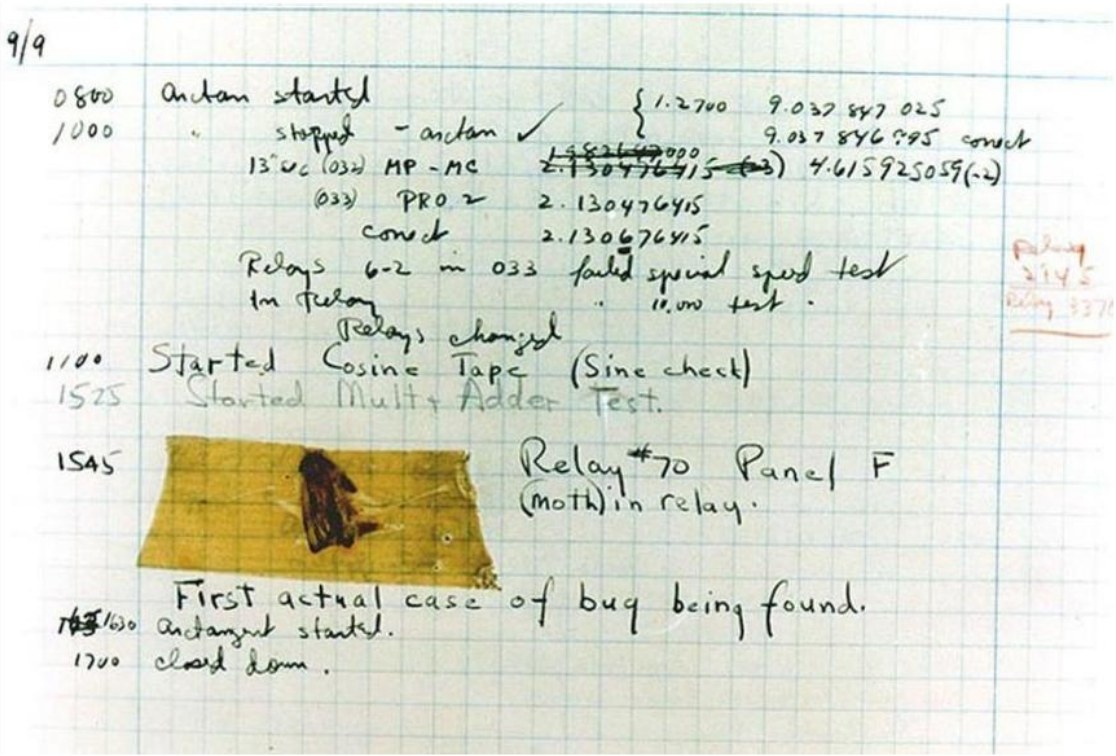
SUBJECTS
English Language Arts, Experiential Learning

PHOTOGRAPH

Computer Bug

"First actual case of bug being found," according to the brainiacs at Harvard, 1945. The engineers who found the moth were the first to literally "debug" a machine.

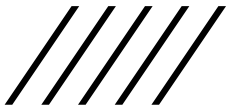
PHOTOGRAPH COURTESY NAVAL SURFACE WARFARE CENTER, DAHLGREN, VIRGINIA



https://education.nationalgeographic.org/resource/worlds-first-computer-bug/

print-statements for å finne feil

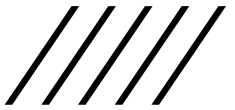
- Ulempe: Må fjernes/skjules etterpå, så brukerne ikke trenger å se masse debugging-informasjon
- Kan likevel være nyttig å lagre debugging-informasjon et annet sted for å fange opp feil ute hos brukeren (som ikke dukket opp under testing)
- Bedre alternativ: logging (til fil eller terminal)
  - Nivå 1: Kun advarsler
  - Nivå 2: Advarsler og informasjon
  - Nivå 3: Advarsler, informasjon og debug-meldinger



LIVEKODING: LOGGING

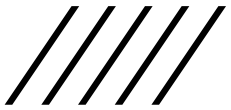
Python Tutor

- God visualisering av (de fleste) objekter, nyttig for å lage en mental modell av hva som skjer i maskinen
- Klarer ikke kjøre “lange” programmer
  - En for-løkke med 2 linjer og 1000 repetisjoner teller som 2000 kodelinjer
- Må ha all koden i samme fil
- Reklame



Debuggeren i VS Code

- Kan pause programmet der vi vil og følge med på hva som skjer med objekter/variabler
- Eventuelt pause bare når en betingelse er oppfylt (for å slippe de 100 første stegene der feil *ikke* skjer)
- Kan velge om vi vil jobbe på høyt nivå (ikke se hva som skjer inni en funksjon som kalles) eller lavt nivå (gå inn i funksjonen)
- Kan endre verdier av variabler for å teste
- Kan printe til egen debug-terminal uten å printe til terminalen som brukeren ser



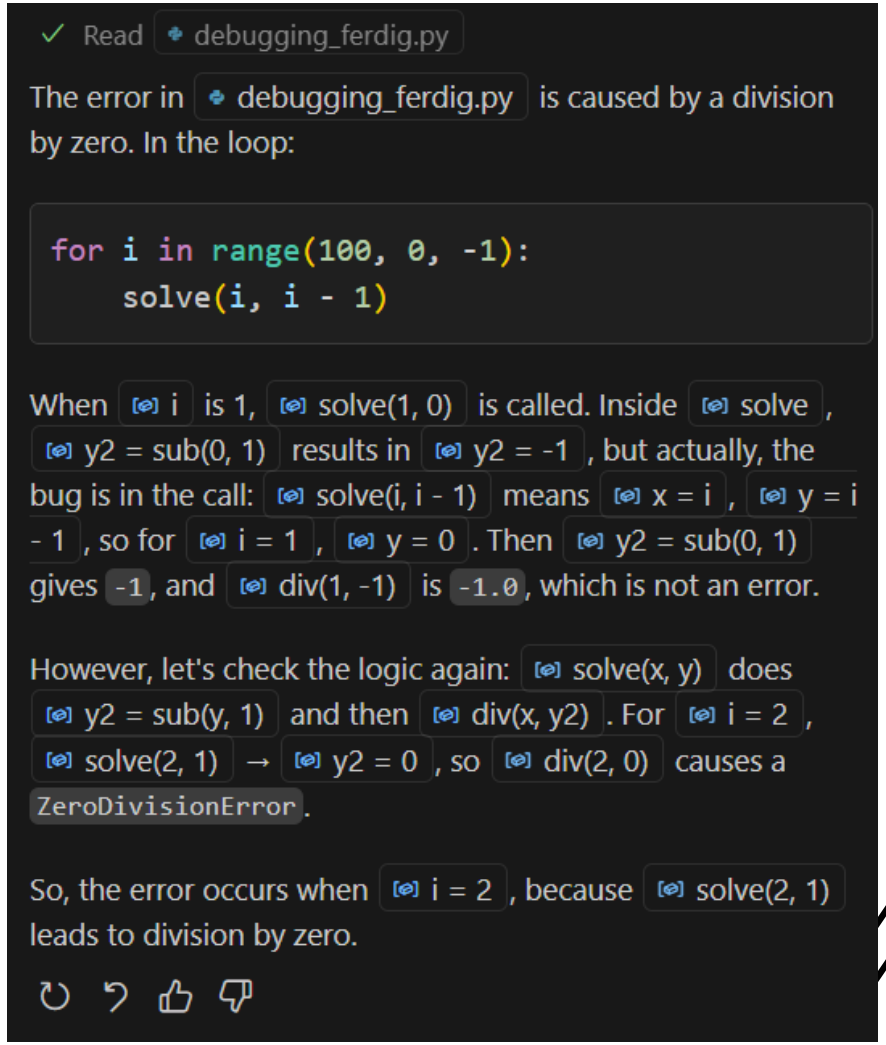
LIVEKODING: DEBUGGING

OBS: Debugging i C++ (på Mac)

- Det finnes en egen oppskrift for dette

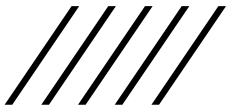
Hva med KI til debugging?

- KI kan brukes hvis du leter etter en feil og står fast og ville spurt en gruppelærer
- Kan lese koden din og gi ideer til hvor du kan lete (selv de gangene den tar feil)
- Men du bør alltid sjekke disse forslagene i debuggeren og ikke stole blindt på KI



Fordeler med debugger

- Læring: Du får en god mental modell av hva som skjer i maskinen når programmet kjører – du kan følge med på alle variabler og objekter (tilstanden til programmet) underveis
- Kontroll: Det er du selv som går gjennom koden på det detaljnivået du ønsker uten å bare måtte stole på at et svar er riktig
- Mestring: Det er gøy å finne ikke-trivielle feil og man vokser på det som programmerer



LIVEKODING: LAG EN BUG

(BILDE: BING IMAGE CREATOR)