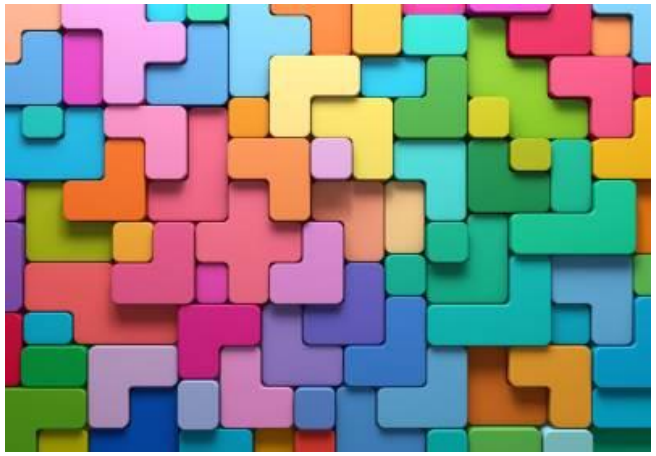
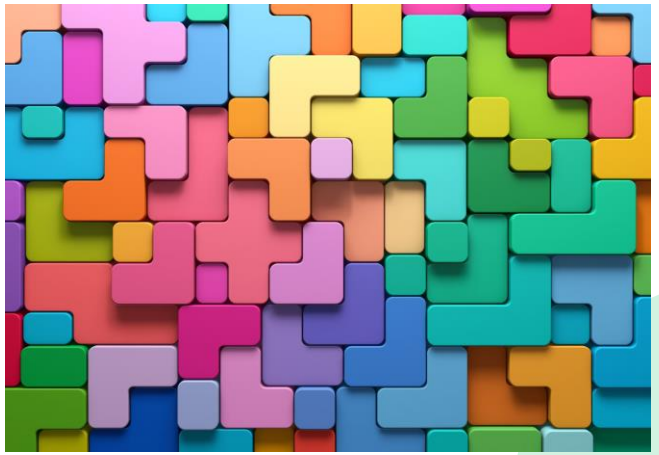
PROFILERING OG OPTIMALISERING

FORELESNING 20

FREDAG 1/11

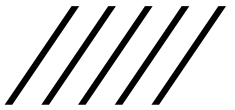




(bilder generert av bing image creator)

Debugging i python (Mac)

- Prøv Python Debugger-extension i VS Code
- (to be continued…?)



RNG seed eller numpy.random seed?

```
seed = [redacted]
rng = np.random.default_rng(seed)
```

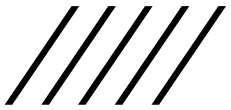
1 per program

```
walkers = MazeWalker(M, maze, rng)
```

```
np.random.seed([redacted])
```

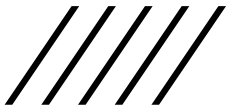
Læremål: Algoritmeanalyse og optimalisering

- Å kunne analysere hvor rask en algoritme er (viktig når vi får enorme mengder data)
- Profilering: finne flaskehalser (10% av koden bruker vanligvis 90% av tiden)
- (+ mer i ukene som kommer)

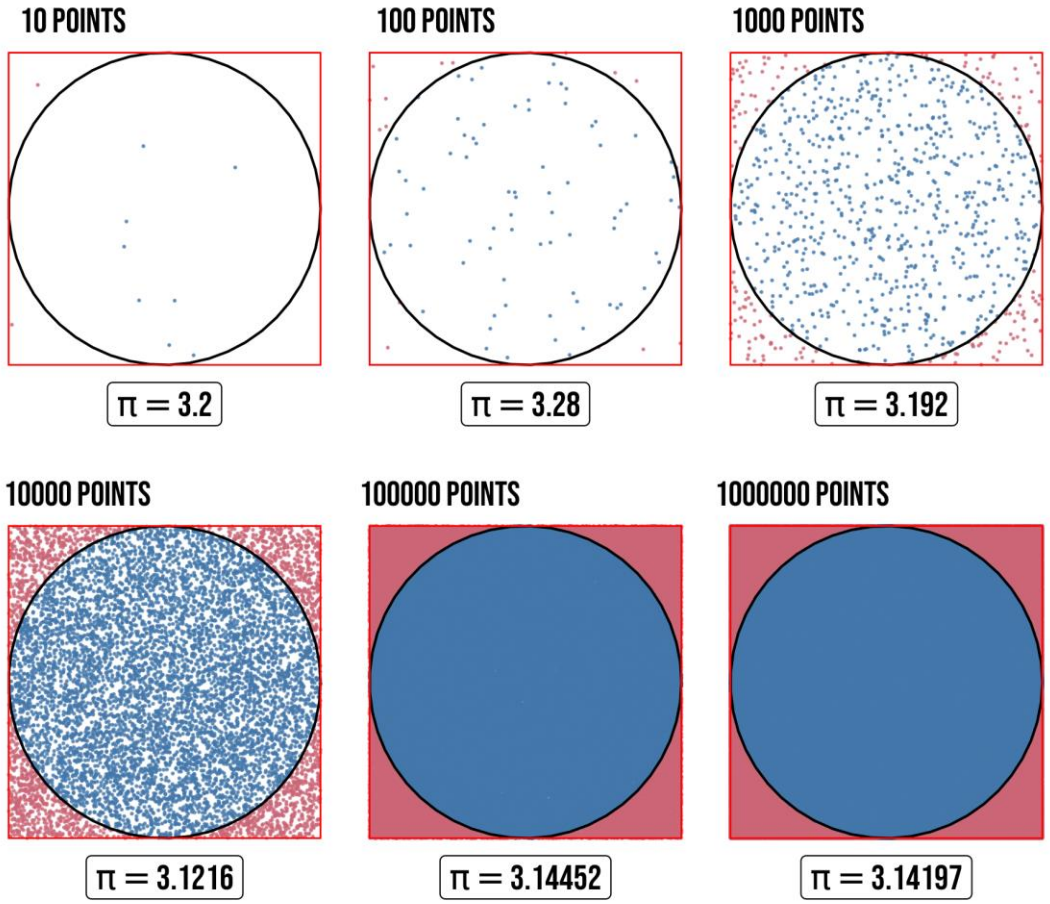


Livekoding over flere forlesninger

- Gjenbruker kode som ligner det vi gjorde i forrige forelesning
- Før du begynner livekodingen:
  - Lag en fork av dette repo'et (public)
  - Direkte link: https://github.uio.no/oddps/IN1910-live/fork



Monte Carlo-integrasjon



https://mauriciocely.github.io/blog/2020/08/05/estimating-pi-using-the-monte-carlo-method/

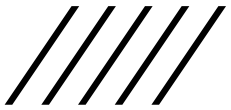
LIVEKODING:

METODER FOR Å FINNE PI

(DEL 2)

Når skal vi optimalisere kode?

1. Få det til å virke
   - Ikke tenk på kjøretid her
   - Test grundig og nøye at det virker
2. Få det til å virke *bra*
   - Gjør koden leselig og enkel å vedlikeholde (elegant kode)
   - Del opp i klasser/funksjoner, lag bedre variabelnavn osv.
3. Få det til å virke bra og *raskt*
   - Finn flaskehalsene og se om de kan gjøres raskere (må testes)
   - Algoritmeanalyse og evt. blandet programmering



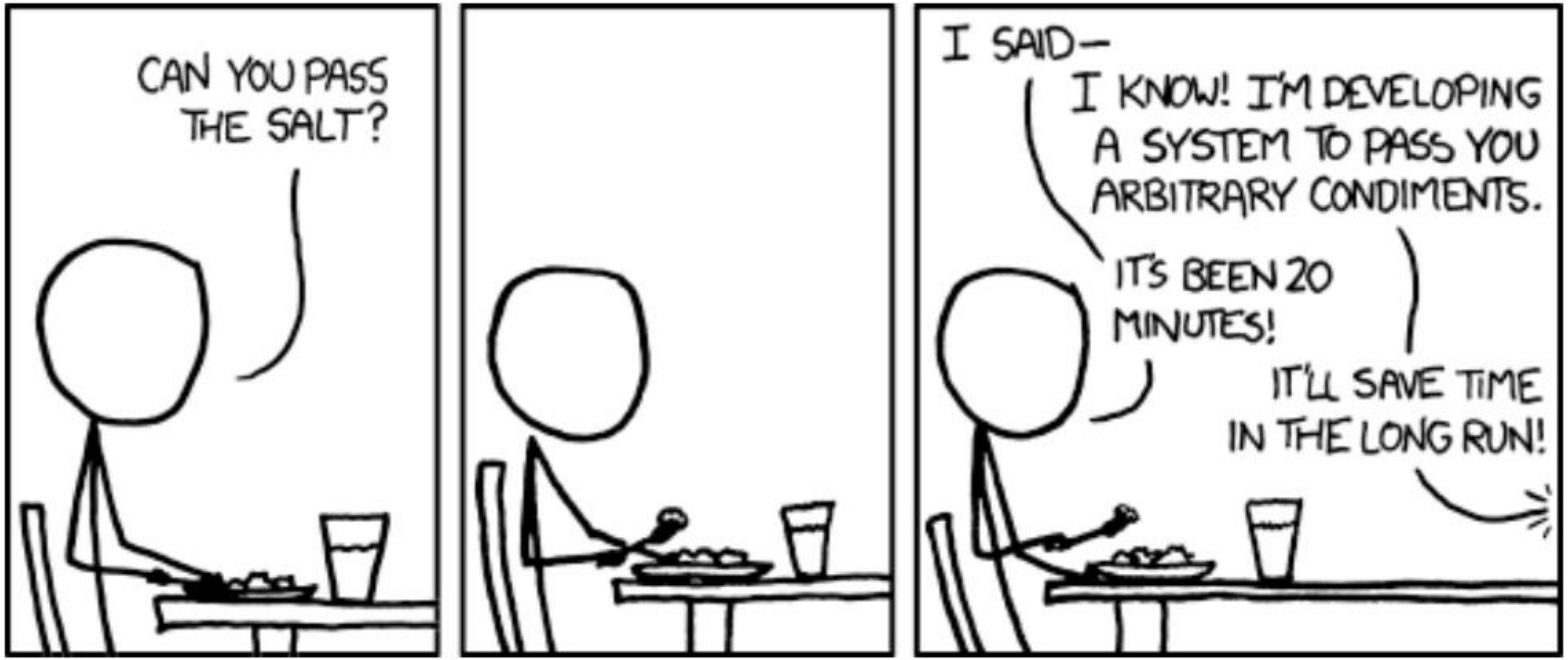


Fig. 59 Source: XKCD #974

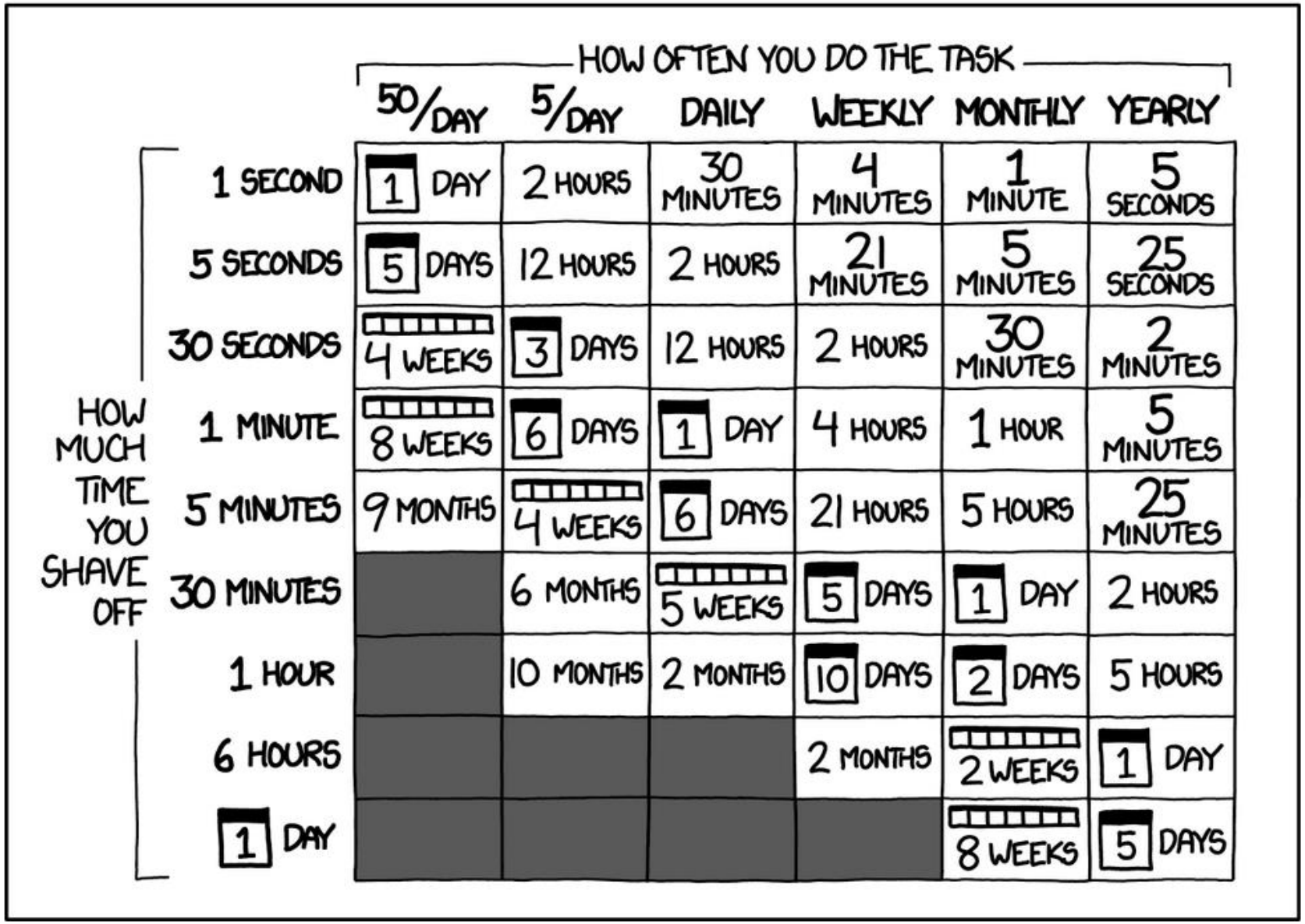
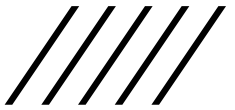


Fig. 58 Source: XKCD #1205

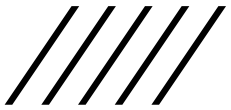
Ulemper med optimalisering

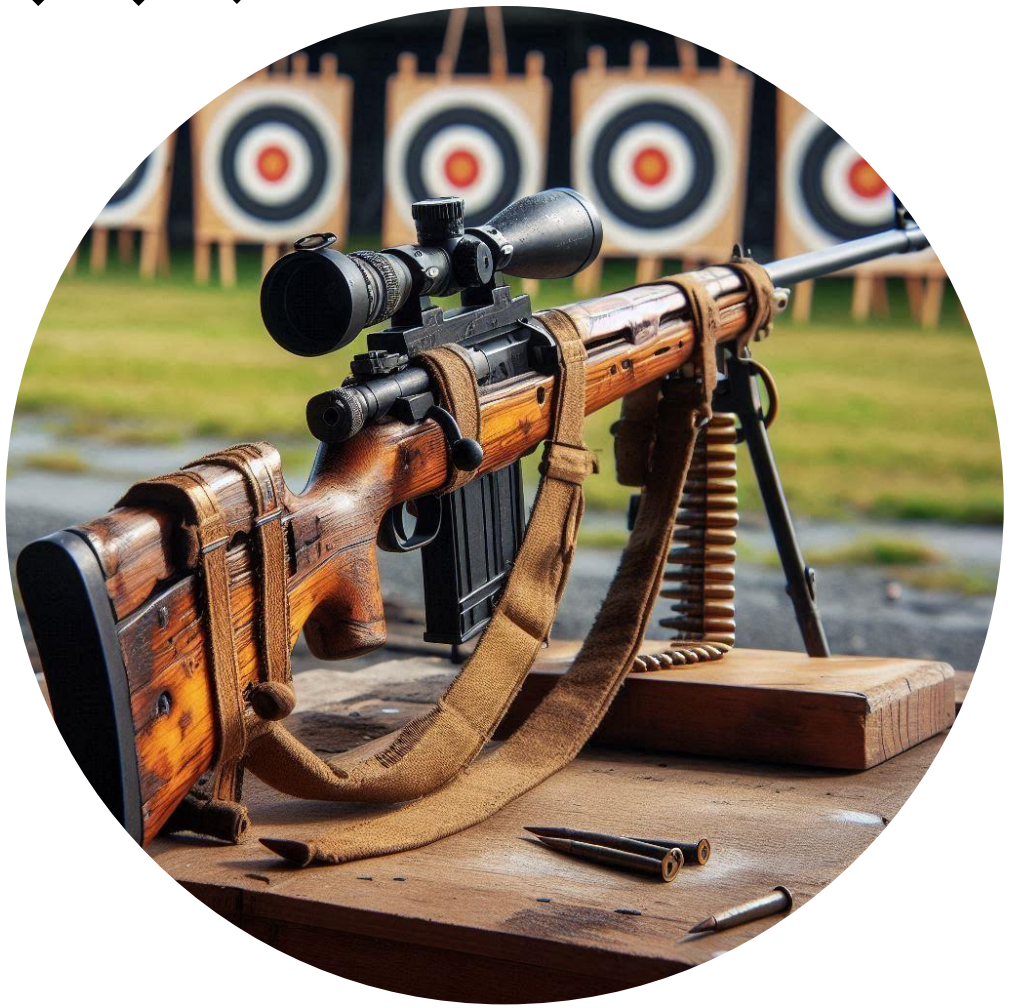
- Tar tid fra andre ting (ny funksjonalitet, bedre testing, osv.)
- Kan gå ut over leseligheten (i noen tilfeller) - hva er viktigst?
- Kan resultere i bugs
- Vi må derfor veie fordelene med raskere kjøretid opp mot disse ulempene



Hvis vi skal optimalisere, hvordan gjør vi det?

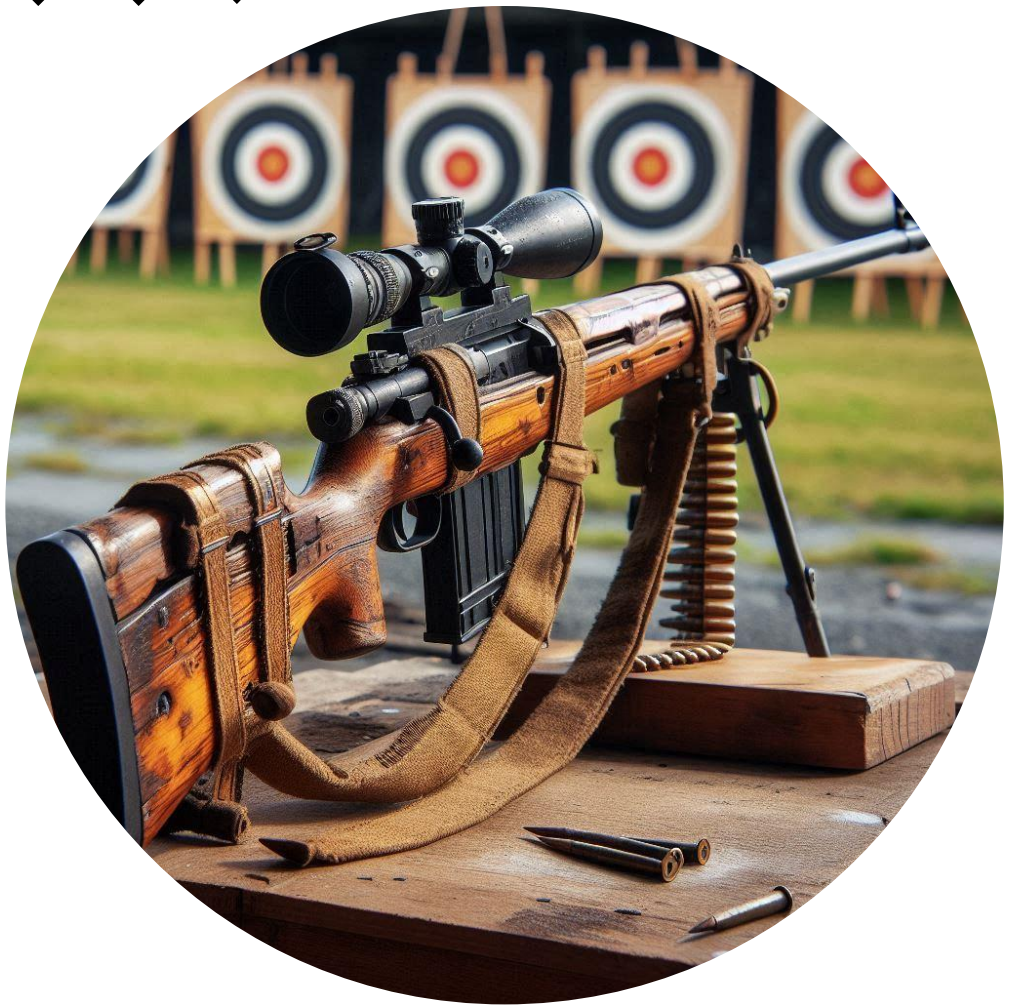
- Analysere algoritmer for å finne den teoretiske forskjellen mellom ulike fremgangsmåter
- Ta tiden på koden (benchmarking) for å finne ut hvor rask den er i praksis
- Profilering for å forstå hvilke deler av koden vi trenger å optimalisere





Benchmarking

- “Hvor nøyaktig er dette geværet?”
- Det varierte med hvor god skytteren (**marks**man) var med akkurat det våpenet, så for å eliminere den variabelen festet man våpen til en **benk** for å kunne sammenligne nøyaktigheten



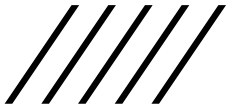
Benchmarking

- I kodesammenheng er det viktig at man da tester ulike versjoner av koden mot *samme* benchmark (og ikke endrer test-caset over tid)
- Kan da teste ulike tilnærminger med samme input og se på kjøretid og nøyaktighet (på output)

LIVEKODING: PROFILERING

cProfile

- **ncalls**: The number of times the function is called
- **tottime**: Time spent inside the function without sub-functions it calls
- **cumtime**: Time spent inside the function *and* inside functions it calls
- **per call**: **cumtime** divided by the number of calls



Etter forelesningen

- Vurderingskriterier til prosjekt 3 er ute
- Devilry-innlevering er også publisert