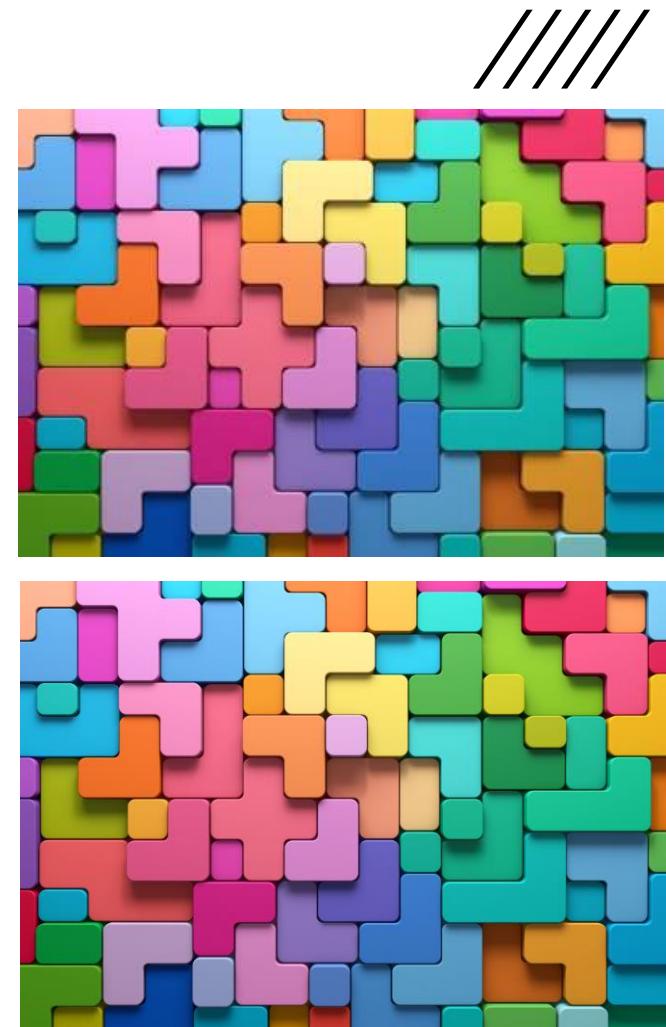


**V E L K O M M E N
T I L
I N 1 9 1 0**

FORELESNING 1
ONSDAG 21/8



○ Infodump?

- Jeg skal prøve å ikke bombe dere med alt for mye informasjon i dag, og heller spre det litt utover de kommende forelesningene etter hvert som det blir relevant
- På fredag kommer mer informasjon om prosjektene



○ Forventninger til dere

- Jeg forventer at dere kommer på forelesning selv om det ikke er obligatorisk
- Jeg forventer at dere deltar i aktivitetene her
- Jeg forventer at dere trives og opplever at det er nyttig at dere kommer hit, både for dere selv og medstudentene
- Jeg forventer at dere kommer på gruppetime for å jobbe med prosjekter og oppgaver, og spør om hjelp når det trengs
- Jeg forventer at dere leverer 100% deres eget arbeid



○ Motivasjon: Hvorfor er vi her?

- Det finnes mange eksempler på forskere som er dyktige i faget sitt
- Og som kan lage programmer som gjør det de skal
- Men programmene kan være vanskelige for andre å lese eller bruke
- Eller vanskelige å endre / bygge på (til og med for oss selv!)
- Vi er her for å lære å skrive vitenskapelige programmer på en måte som gjør disse tingene lette for både andre og oss selv!





Best Practices for Scientific Computing

1. Skrive programmer så mennesker forstår dem, ikke bare datamaskinen
2. La datamaskinen gjøre det datamaskinen er god til (repetere, automatisere)
3. Gjøre endringer i små steg av gangen (og kunne gå tilbake hvis det oppstår problemer)
4. Unngå å gjenta det du selv (eller andre) har gjort før





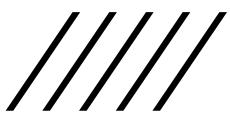
Best Practices for Scientific Computing

5. Planlegge for at det oppstår feil (testing, debugging)
6. Gjøre programmet raskt først når det virker som det skal
7. Dokumentere design og formål, ikke tekniske finurligheter
(*hva* koden gjør, ikke *hvordan* den gjør det)
8. Samarbeide for å finne feil og holde orden på dem





Læremål





Forkunnskaper

- Vi antar du har brukt Python før, men ikke nødvendigvis på en stund
- Eventuelt at du har programmert i et annet programmeringsspråk
- For dere som ikke har brukt Python før eller på en stund, anbefaler vi å bruke ekstra tid på emnet, slik at du får repetert det grunnleggende i Python
- (Hvis du *aldri* har programmert før, er IN1900 eller IN1000 et mye, mye bedre sted å starte!)



○ Hva er pensum?

- Forelesningene: se [timeplanen på emnesiden](#)
- Alle de fire prosjektene



○ Fem ulike måter å forstå på

- Å kunne **forklare** til noen andre hvordan noe virker (dokumentasjon i prosjektene, muntlig eksamen)
- Å kunne **tolke** seg frem til hva som er relevant akkurat her (prosjektene, muntlig eksamen)
- Å kunne **bruke** det du har lært (prosjektene)
- Å kunne se ting fra flere **perspektiv**: utvikler, tester, bruker (prosjektene, muntlig eksamen)
- Å forstå hva du forstår og ikke forstår (**metakognisjon**) (muntlig eksamen)





Vurdering i IN1910

Mappeevaluering samt muntlig eksamen. Studentene utvikler og leverer et større programsystem med tilhørende dokumentasjon, og alle vil bli kalt inn til en muntlig høring hvor de må demonstrere at programmet virker og forklare sentrale deler av programmet og den underliggende matematikken.

Alle innleveringene må være godkjente for å få bestått karakter i emnet. Både mappen og den muntlige eksamenen må være bestått for å få bestått karakter.

Bestått/ikkje bestått

Karakterskalaen med bestått og ikkje bestått er eit sjølvstendig vurderingsuttrykk. Karakterskalaen er utan samanheng med bokstavkarakterane, og karakterane blir ikkje konvertert i samband med godkjenning.





Kommunikasjon

- Vi bruker [Mattermost](#) til kommunikasjon utenom forelesning og gruppertimer (logg inn med UiO-brukernavn og passord)
OBS: Krever at du setter opp [tofaktor-autentisering](#)
- Du skal automatisk bli lagt til IN1910-gruppen
- Her er emneansvarlig, gruppelærere og andre studenter
- Her kan man ønske seg ting forklart til kommende gruppertimer og forelesninger





Emnesiden

- All informasjon blir lagt ut her, og vi forventer at dere til enhver tid følger med på det som blir publisert (og eventuelt endret) gjennom hele semesteret
- Timeplan (med forelesningsnotater og opptak)
- Beskjeder (med info om prosjektene og eksamen)
- Ukesoppgaver og andre ressurser





Litt mer om timeplanen

- Tema for de første ukene er klare, timeplanen oppdateres etter hvert
- Det er lagt inn ekstra tid underveis i tilfelle vi trenger det
- Hvis ikke, vil denne tiden bli frigitt til ekstra arbeid med prosjekter uten veiledning (her i auditoriet eller hvor dere vil)
- Følg derfor godt med på timeplanen for å få med deg når dette eventuelt skjer
- Gruppetime starter først neste uke (mandag 26/8 →)



○ Hva slags programvare trenger jeg?

- Python-distribusjon: [Anaconda](#)
- Editor: [Visual Studio Code](#) (VS Code)
(eller en annen en du liker – men da får du ikke sett på forelesning hvordan man gjør ting direkte i din editor)
- Terminal: PowerShell / bash
Dette har du allerede på maskinen!
- [GitHub Desktop](#)
- (Senere i semesteret: En kompilator til C++)



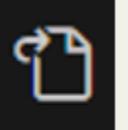
Live-koding

- Det vi skal gjøre etter pausen
 - Vi skriver alle den samme koden, jeg på storskjerm og du på din egen maskin (så du har din egen kopi av koden du kan beholde etterpå)
 - Hvis du står fast, får feilmelding eller lignende:Se på opptaket og finn feilen senere - det er OK å gå videre selv om det ikke funker, ikke bruk tid på å finne feilen her og nå
 - Rekk opp en hånd hvis du synes det går for fort!
 - (Bruk opptaket til å livekode senere hvis du er borte en dag) //



Pause

- Etter pausen skal vi repetere litt Python (live-koding)
- For å komme inn i en god vane med å bruke terminalen, anbefales det å legge inn følgende innstillinger i VS Code:

Windows: File → Preferences → Settings → 
Mac: Code → Settings → 

```
"terminal.external.windowsExec": "C:\\\\WINDOWS\\\\system32\\\\WindowsPowerShell\\\\v1.0\\\\powershell.exe",  
"terminal.explorerKind": "external",  
"python.terminal.executeInFileDir": true,
```



• Litt om denne repetisjonen

- Dette er *ikke* en fullverdig introduksjon til Python
- Friske opp det dere har lært som kanskje ikke har vært brukt på en stund





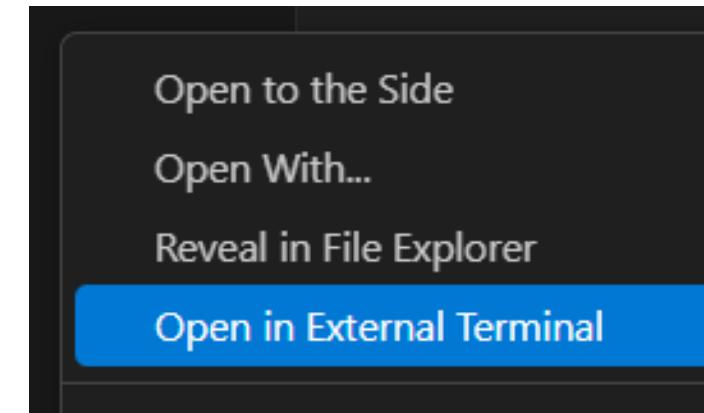
Live-koding

- Hvis du nå har en editor (f.eks. VS Code) og Python (helst Anaconda, som du uansett trenger snart), bli med på live-kodingen
- Hvis du mangler editor / Python anbefales det å gå gjennom opptaket i ettertid så du får prøvd live-kodingen selv
- Og du bør få editor og Python på plass så fort du kan
- Generelt når vi live-koder vil mye av det vi gjør ikke være med på lysarkene fra forelesningen (se heller på opptak i disse tilfellene)



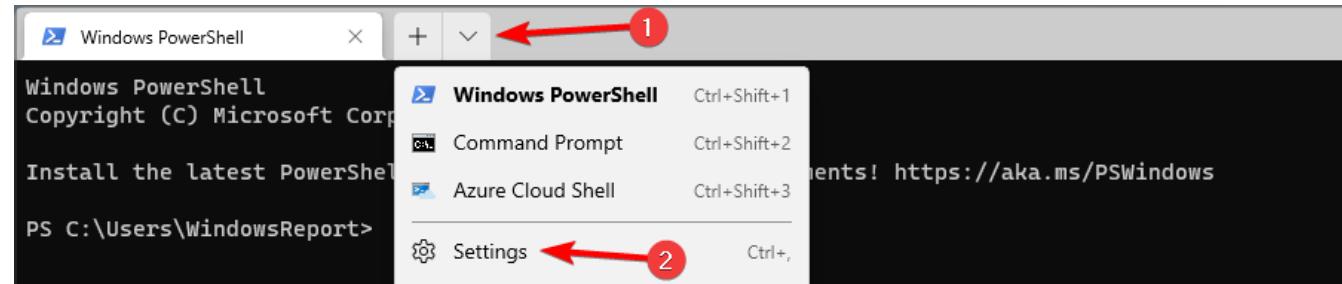
“Hvorfor bruker du ikke play-knappen?”

- Fordi det er en dårlig vane å teste programmet i editoren du bruker til å skrive koden med
- Brukeren av programmet ditt skal bare trenge en terminal og de riktige bibliotekene
- Det hjelper lite å si “ja, men det virker jo på min maskin” hvis brukeren (eller retteren) ikke klarer å kjøre koden din
- Derfor: Bruk terminalen til kjøring og testing!

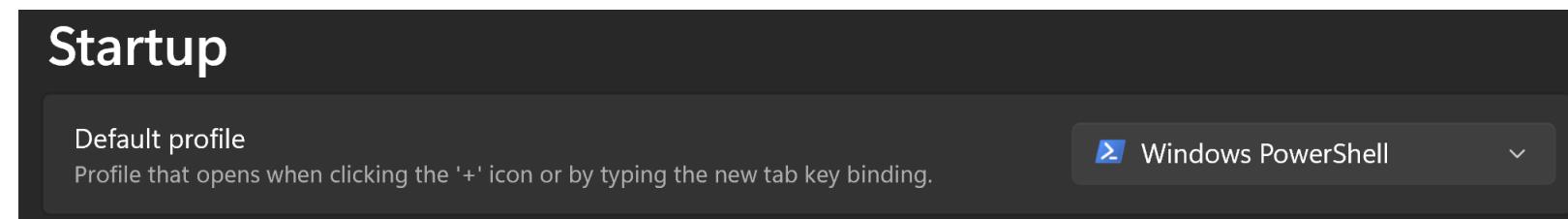


• Hvis du bruker Windows...

- Står det Windows PowerShell øverst i vinduet?
- Hvis nei, trykk på:



- Velg Windows PowerShell som standard, lukk terminalvinduet og åpne det på nytt



○ Anaconda i terminalen

- Ser du (base) i terminalvinduet?
- Hvis ikke
- PowerShell (administrator):
set-executionpolicy remotesigned
- Anaconda Prompt:
conda init powershell (Windows)
conda init (Mac/Linux)
- Åpne nytt terminalvindu fra VS Code





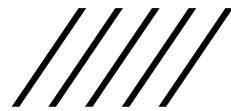
Noen begreper

- *Variabler* har verdier
- Alle verdier i Python er *objekter* som finnes i minnet til datamaskinen et sted
- Så en variabel er et navn som vi gir til et objekt
- Du har egentlig brukt objekter (verdier) i Python fra dag 1
- Objekter kommer i ulike *typer* (**int, str, float, list...**)
- Programkoden som beskriver hva en type objekt kan gjøre (og hvordan) kaller vi for en *klasse*



Ikke-muterbare typer

- Objekter av en *ikke-muterbar* type (eller klasse) kan ikke endres
- Når jeg har definert **a = 4** kan jeg ikke endre verdien av 4 etterpå
- Hvis jeg etterpå skriver **a = 5** har jeg ikke endret 4, dette altså et *annet* objekt (som ligger et annet sted i minnet)
- Jeg kan endre hvilket objekt variabelen **a** står for, men selve objektene 4 og 5 endres ikke
- Eksempler: **bool, int, float, str, tuple**
(en **tuple** er liste som ikke kan endres)





Muterbare typer

- Objekter av en *muterbar* type (eller klasse) kan derimot endres underveis i programmet
- Når jeg har definert **a = []** er listen tom til å begynne med
- Hvis jeg etterpå skriver **a.append(1)** er det fortsatt det samme objektet (på samme sted i minnet), men det er endret fra **[]** til **[1]**
- Eksempler: **list**, **dict**, **set**, ...
- “Men har alt dette noe å si i praksis, da...?”



○ Noen flere begreper

```
1  def f(x):  
2      |     return x**2  
3  
4  y = f(2)
```

Parameter	x
Argument (inndata)	2
Returverdi (utdata)	4

2 → f(x) → 4



● while-løkker

- mest generell, kan brukes i alle situasjoner (selv når vi ikke vet antall repetisjoner på forhånd)
- kan lett gå uendelig om vi ikke passer på

while <en test returnerer True>:

<kode som repeteres>

<kode som repeteres>

<kode som ikke repeteres (kjører etter at løkken er ferdig)>



for-løkker

- mer spesialisert løkke, krever en *samling* av objekter
(list, tuple, dict, range...)
- går gjennom alle disse objektene (*elementene*) i samlingen

for <element> **in** <samling>:

<kode som repeteres>

<kode som repeteres>

<kode som ikke repeteres (kjører etter at løkken er ferdig)>



• Bygge en liste med løkke

[<uttrykk> **for** <element> **in** <samling>]

er en snarvei for å bygge en liste på denne måten:

```
<liste> = []
```

```
for <element> in <samling>:
```

```
    <liste>.append(<uttrykk>)
```



○ Hente indeks og element samtidig

for <indeks>,<element> **in enumerate**(<samling>):
 <gjør noe med indeks og element>

er en snarvei for:

for <indeks> **in range(len(<samling>)):**
 <element> = <samling>[<indeks>]
 <gjør noe med indeks og element>



• Gå gjennom *flere* samlinger på en gang

for <element1>,<element2> **in zip**(<samling1>, <samling2>):
 <gjør noe med de to elementene>

[1, 2, 3] ["a", "b", "c"]



- Lager en liste med **tupler**
- Kan også gjøres med 3 eller flere samlinger
- Alle samlingene bør ha like mange elementer

[(1, "a"),
(2, "b"),
(3, "c")]



Matematisk eksempel: Forward Euler

$$\frac{du}{dt} = -au(t)$$

$$\frac{\Delta u}{\Delta t} = -au(t)$$

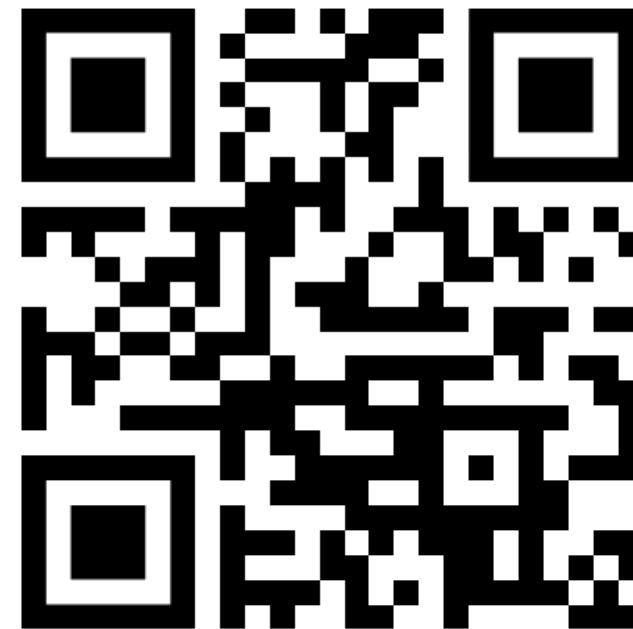
$$\frac{u_{i+1} - u_i}{\Delta t} = -au_i$$

$$u_{i+1} = (1 - a\Delta t)u_i$$



• Tilbakemelding på dagens forelesning

- <https://nettskjema.no/a/440125>
- Du kan også svare dersom du har sett forelesningen i opptak





Etter forelesningen

- Det vi ikke rakk å live-kode i dag kan du selv friske opp [her](#)
- På fredag begynner vi å se på versjonskontroll med GitHub
- Da gjør vi klart alt til prosjekt 0 her i forelesningen
(og dere får mer info om prosjektet)
- Imens, husk å installere det du eventuelt mangler:
 - [Anaconda](#)
 - editor (gjerne [VS Code](#), da får du mye gratis i forelesningene)
 - [GitHub Desktop](#) (Windows/Mac)
 - Linux: Enten følg [denne oppskriften](#) eller bruk [git i terminalen](#) istedenfor

