

Undervisning med notebooks

– tanker, tips og refleksjon

Del 2: Barekratisk vekst
I del 1 så vi at vår eksponentielle vekst modell fører til 2.5 millioner kaniner på øya etter 100 år. Men denne modellen vil raskt ikke passe til realiteten.

Problemet er at matematisk modellen vi har laget så langt har ikke hatt form for overvurdering. Dvs. vi har ikke tatt med faktorer som viser at veksten ikke er basert på hvor mange kaniner som er på øya fra før. Vi ønsker en modell på former:

$$k_{t+1} = k_t + r(k_t)k_t.$$

Det rettlinige spørsmålet vi har i hvilken form vi tror at $r(k)$ skal ha. La oss starte med noen obserasjoner:

1. Dya har en bølgende, N, som er antallet kaniner gya har plass og må til.
2. Giv $k_t > 0$ så ønsker vi eksponentiell vekst for de er bærekunnen ikke hoyere enn antall dyr.
3. \bullet Niv $k_t = N$, så ønsker vi at det skal være ca like mange dyr på øya hvert år.
4. \bullet Antall er $r(N) = 0$.

Den letteste måten å få en funksjon på denne formen er å tenke seg en rett linje som går gjennom punktene $(0, r(0))$ og $(N, 0)$, slik vi ser i figuren under.

Hvis vi bruker funksjonen over for den relative vekstraten at vi vi dette uttrykket for r :

$$r(k) = r_0 \left(1 - \frac{k}{N}\right).$$

Opgave 1 a)
Lag en funksjon vekstrate(`antall_kaniner`) som tar inn antall kaniner og returnerer vekstraten for et system barevenn med $N = 5000$ og ubegrensete vekstrate $r_0 = 0.5$.



Tekna

kodeskolen

simula

“Shift-Enter” tekstdokument

Eksempel:

Kode for å simulere dartkast

Når vi kaster en darpil så lander den ett eller annet tilfeldig sted på enhetskvadratet. For å simulere dette kan vi lage en funksjon som trekker et x-koordinat tilfeldig mellom -1 og 1 og et y-koordinat tilfeldig mellom -1 og 1. Dette kan vi gjøre med `uniform`-funksjonen i pylab

```
In [ ]: def kast_dart():
    """Kast en tilfeldig dart og returner koordinatene til kastet (x og
    """
    x_koordinat = uniform(-1, 1)
    y_koordinat = uniform(-1, 1)

    return x_koordinat, y_koordinat
```

Visualisere et dartkast

Det er ofte nyttig å teste koden underveis og visualisere koderesultatene. Under er litt kode for å plotte hvor darkastet vårt lander inne i enhetskvadratet.

```
In [ ]: dart_x, dart_y = kast_dart()

plot(dart_x, dart_y, 'x')
axis("square") # Pass på at x-aksen og y-aksen har samme målestokk
xlim(-1, 1) # Pass på at x-aksen går fra -1 til 1 (bredden til enhetskv
ylim(-1, 1) # Pass på at y-aksen går fra -1 til 1 (høyden til enhetskv
show()
```

Fordeler:

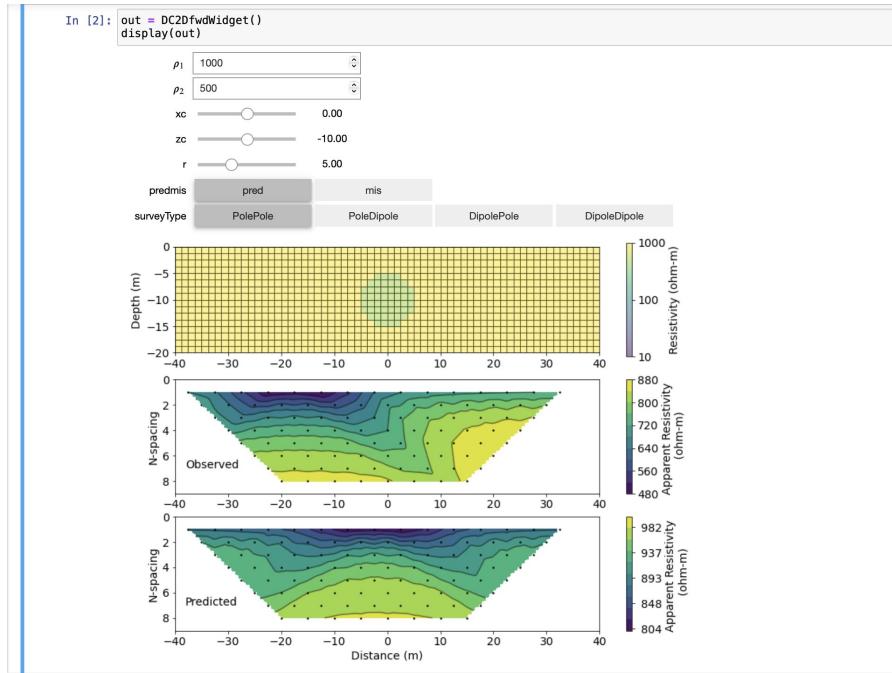
- Kan gi en grundig oversikt over et tema
- Mer interaktivt enn statisk dokument/lærebok

Ulemper:

- Er ikke veldig interaktivt
- Kan bli til at elever trykker shift enter for å komme videre uten å se på hva som kjøres

Notebook som et digitalt verktøy

Eksempel:



Fordeler:

- Oppfordrer til utforskning
- Gjør det lett å teste hypoteser selv og få øyeblikkelig tilbakemelding

Ulemper:

- Krever strukturerte spørsmål for elevene å svare på

Fikle notebook

Eksempel:

Modifiser dødsraten og fødselsraten i cellen under og se hvordan kaninutviklingen endrer seg

```
In [9]: antall_år = 10  
  
dødsrate = 0.1  
fødselsrate = 0.2  
vekstrate = fødselsrate - dødsrate # regne ut vekstraten  
  
k = zeros(antall_år)  
k[0] = 200  
  
for t in range(antall_år-1):  
    k[t+1] = k[t] + vekstrate*k[t]  
  
plot(range(antall_år), k, '-o')  
xlabel('Tid [år]')  
ylabel('Antall kaniner')  
show()
```



Fordeler:

- Fin måte å bli vant til kode
- Rask "seier"!

Ulemper:

- Gir liten eierskap
- Trenger liten forståelse av problemet og løsningene

“Fyll inn det som mangler” notebook

Eksempel:

$$k_{t+1} = k_t + 100,$$

hvor k_t er antall kaniner etter t år. Hvis vi vil skrive det med Python, så skriver vi

```
k[t+1] = k[t] + 100
```

Det er altså nesten likt den matematiske formen!

Fyll inn den manglende linja i kodeblokka under.



```
In [ ]: from pylab import zeros # Importere zeros for å lage en "tom array" med nuller
antall_år = 10

k = zeros(antall_år)
print("Tom array til å begynne med", k)

k[0] = 200

for t in range(antall_år-1):
    k[t+1] = ...

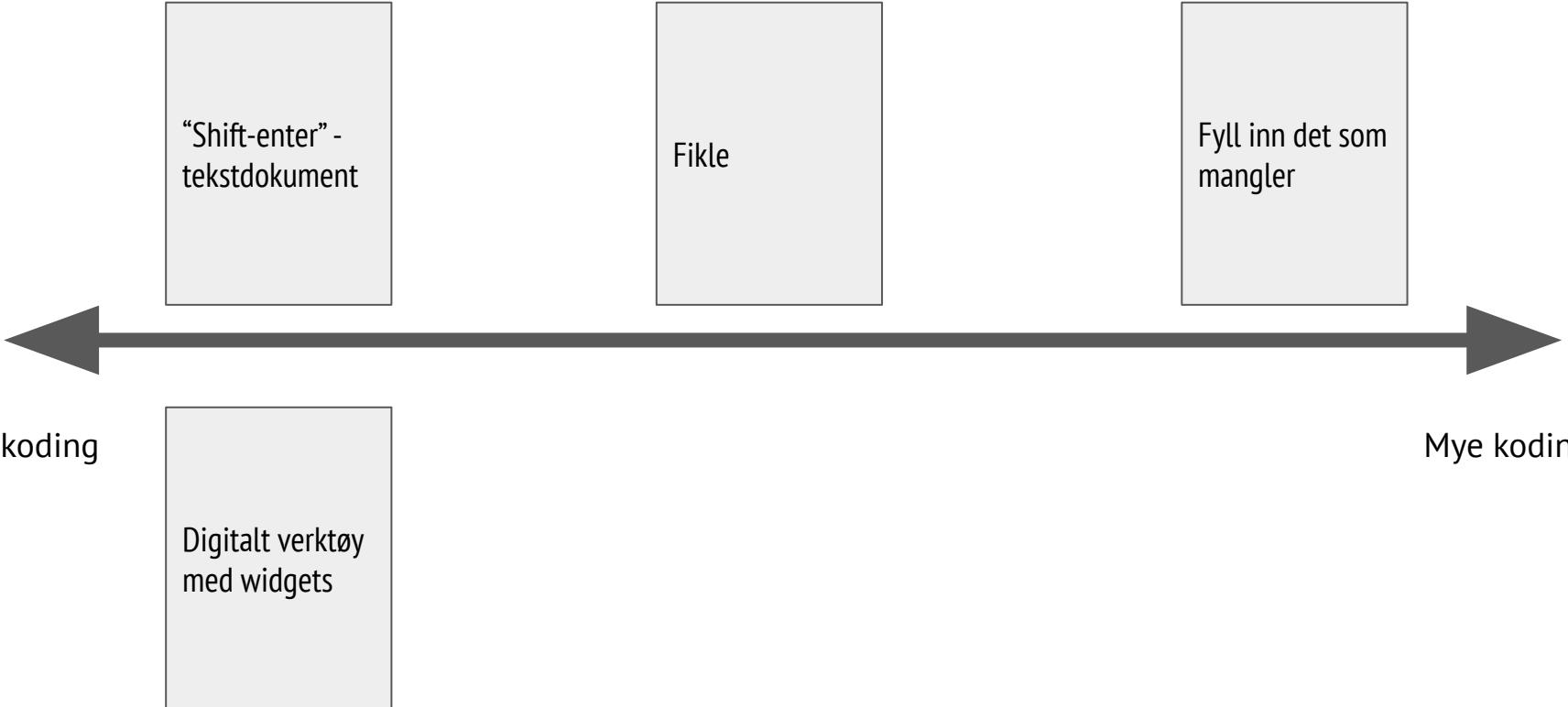
print("Array med simuleringsresultater", k)
```

Fordeler:

- Trene på kun det viktige
- Rask tilbakemelding om koden stemmer

Ulemper:

- Får mindre trening i å se helheten i koden
- Kan bli for enkelt



**Skriv i chatten: muligheter og utfordringer med
notebook-dokumenter for din undervisning**

Mulighet: Du kan lage interaktive dokumenter med en blanding av tekst, kode og utskrift

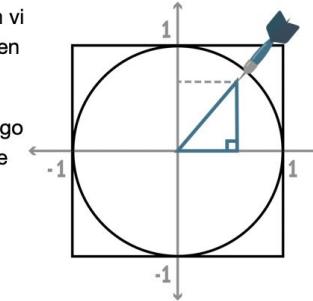
Sjekke at vi treffer blinken

Neste steg i simuleringen vår blir å sjekke om vi traff blinken. Vi trenger altså å sjekke om vi er innenfor enhetssirkelen. For at et punkt skal være innenfor enhetssirkelen må avstanden mellom origo og punktet være mindre enn 1.

Vi ser av figuren til høyre at vi kan tegne opp en rettvinklet trekant slik at avstanden til origo er gitt ved hypotenusen til trekanten og x- og y-koordinatene til dørkastet tilsvarer første og andre katet. Da kan vi bruke Pythagoras-setning til å finne avstanden.

Vi starter med å lage en funksjon for å regne ut hypotenusen til en rettvinklet trekant.
Pythagoras setning sier at:

$$\text{Hypotenus} = \sqrt{\text{Katet}_1^2 + \text{Katet}_2^2}$$



Dette kan vi skrive som en funksjon i Python:

```
In [4]: def finn_hypotenus(katet1, katet2):
    """Tar inn katetene i en rettvinklet trekant og returnerer lengden til hypotenus
    """
    return sqrt(katet1**2 + katet2**2)
```

Nå kan vi bruke `finn_hypotenus` funksjonen til å lage en betingelse for om darten traff inne i enhetssirkelen. Dette kan vi bruke til å lage en funksjon som sjekker om vi traff eller ikke:

Mulighet: Du kan lettere gi delvis ferdig kode med tydelig tekstforklaring

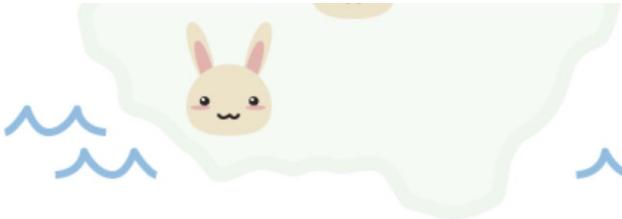
$$k_{t+1} = k_t + 100,$$

hvor k_t er antall kaniner etter t år. Hvis vi vil skrive det med Python, så skriver vi

```
k[t+1] = k[t] + 100
```

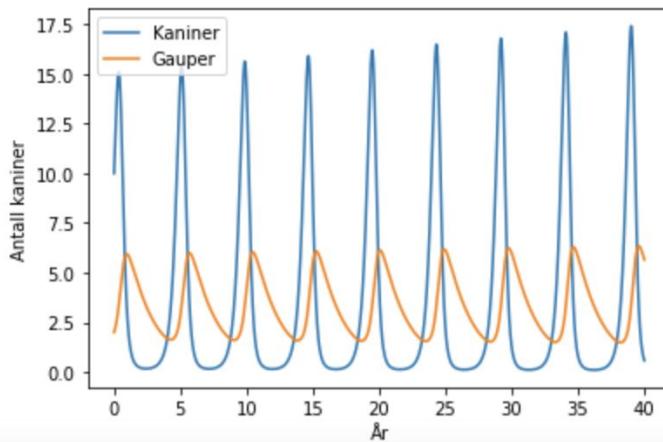
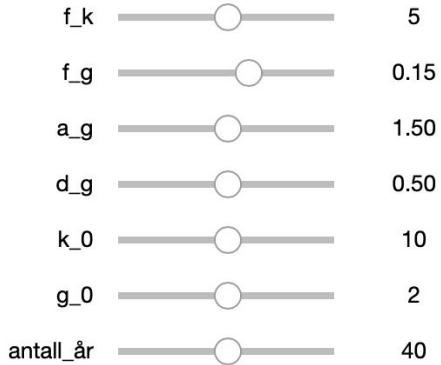
Det er altså nesten likt den matematiske formen!

Fyll inn den manglende linja i kodeblokka under.

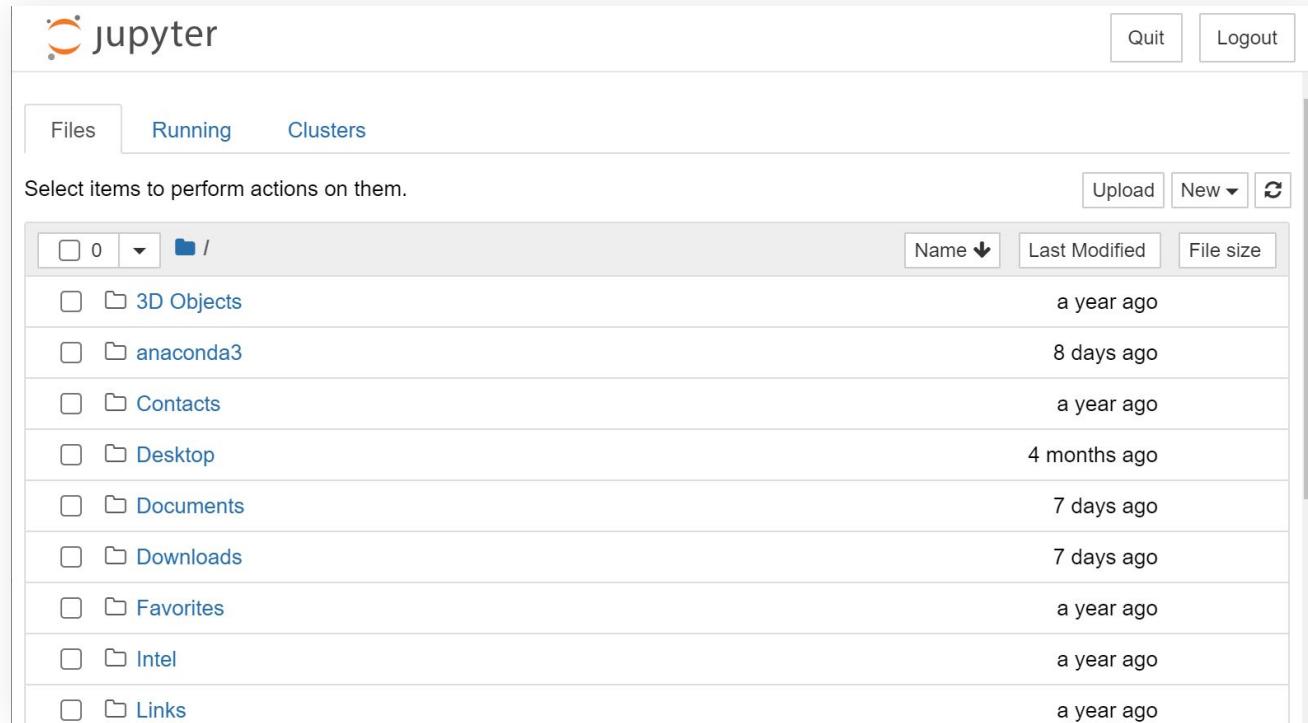


```
In [ ]: from pylab import zeros # Importere zeros for å lage en "tom array" med nuller  
antall_år = 10  
  
k = zeros(antall_år)  
print("Tom array til å begynne med", k)  
  
k[0] = 200  
  
for t in range(antall_år-1):  
    k[t+1] = ...  
  
print("Array med simuleringsresultater", k)
```

Mulighet: Widgets åpner for interaksjon og kan oppfordre til utforskning



Utfordring: Å åpne en notebook krever forståelse for filsystemet på PCen



Utfordring: Det er lett å få uforklarlige feil hvis man kjører cellene i feil rekkefølge

```
In [3]: x = 7
```

```
In [2]: print(x)
```

```
5
```

Utfordring: Det kan være utfordrende å live-kode med notebooks

```
return x_koordinat, y_koordinat
```

Visualisere et dartkast

Det er ofte nyttig å teste koden underveis og visualisere koderesultatene. Under er litt kode for å plotte hvor dartkastet vårt lander inne i enhetskvadratet.

```
In [ ]: dart_x, dart_y = kast_dart()

plot(dart_x, dart_y, 'x')
axis("square") # Pass på at x-aksen og y-aksen har samme målestokk
xlim(-1, 1) # Pass på at x-aksen går fra -1 til 1 (bredden til enhetskvadratet)
ylim(-1, 1) # Pass på at y-aksen går fra -1 til 1 (høyden til enhetskvadratet)

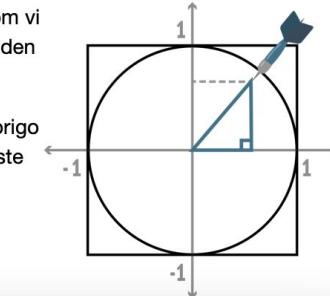
show()
```

Sjekke at vi treffer blinken

Neste steg i simuleringen vår blir å sjekke om vi traff blinken. Vi trenger altså å sjekke om vi er innenfor enhetssirkelen. For at et punkt skal være innenfor enhetssirkelen må avstanden mellom origo og punktet være mindre enn 1.

Vi ser av figuren til høyre at vi kan tegne opp en rettvinklet trekant slik at avstanden til origo er gitt ved hypotenusen til trekanten og x- og y-koordinatene til dartkastet tilsvarer første og andre katet. Da kan vi bruke Pythagoras-setning til å finne avstanden.

Vi starter med å lage en funksjon for å regne ut hypotenusen til en rettvinklet trekant.
Pythagoras setning sier at:



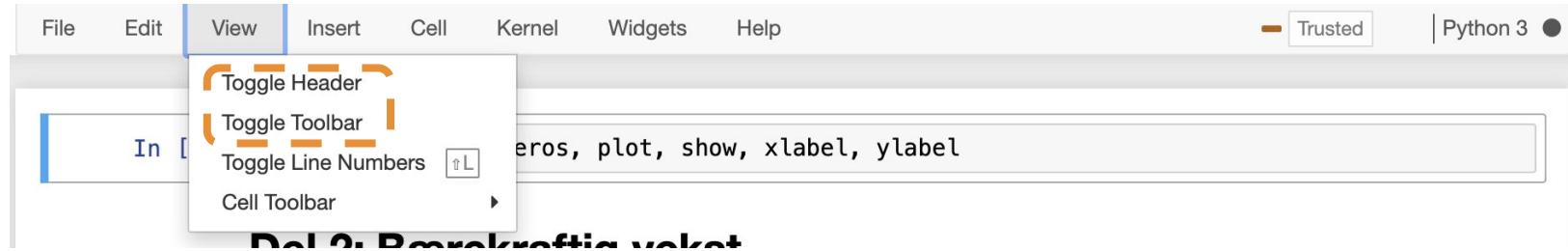
Tips og triks med Jupyter notebooks



Hvis du trykker på *Tab* knappen mens du skriver, vil Jupyter gjette hva skriver



Du kan skjule overskriften og verktøylinja



Du kan bruke notebooks for å generere slides

A screenshot of a Jupyter Notebook interface. On the left, there's a code cell labeled "In [3]:" containing Python code to generate an array and print its second element. The output shows "2" and "3". Above the code cell, a "Cell Toolbar" dropdown menu is open, showing options like "None", "Edit Metadata", "Raw Cell Format", "Slideshow", "Attachments", "Tags", and "Create Assignment". A large grey arrow points from this cell towards the right panel.

```
from pylab import arange
x = arange(5)
print(x[2])
x[2] = 3
print(x[2])
```

2
3

Vi kan indeksere arrayer for å hente ut eller oppdatere elementer

A diagram showing a horizontal array of nine blue squares, each labeled with an index: s[0], s[1], s[2], s[3], s[4], s[5], s[6], s[7], and s[8].

A screenshot of a Jupyter Notebook cell titled "In [3]:". It contains Python code demonstrating array indexing and assignment. The code imports arange from pylab, creates an array x of length 5, prints its second element, changes the third element to 3, and prints it again. The output shows the original value "2" followed by the modified value "3".

```
from pylab import arange
x = arange(5)
print(x[2])
x[2] = 3
print(x[2])
```

2
3

Med RISE-pluginen kan du generere interaktive slides du kan skrive å kjøre kode i



Vi kan indeksere arrayer for å hente ut eller oppdatere elementer

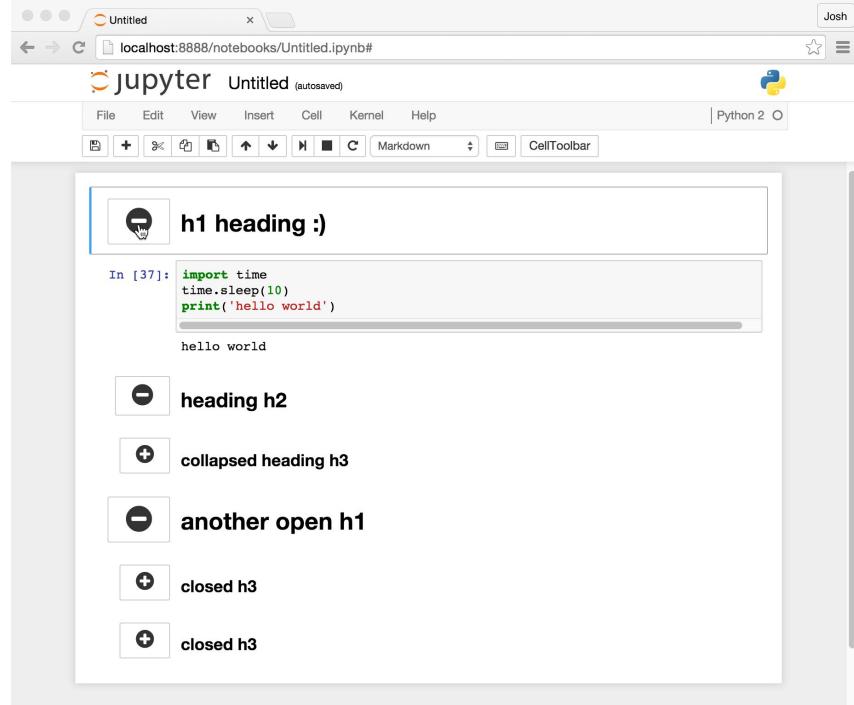


```
In [3]: from pylab import arange  
  
x = arange(5)  
print(x[2])  
x[2] = 3  
print(x[2])
```

2
3

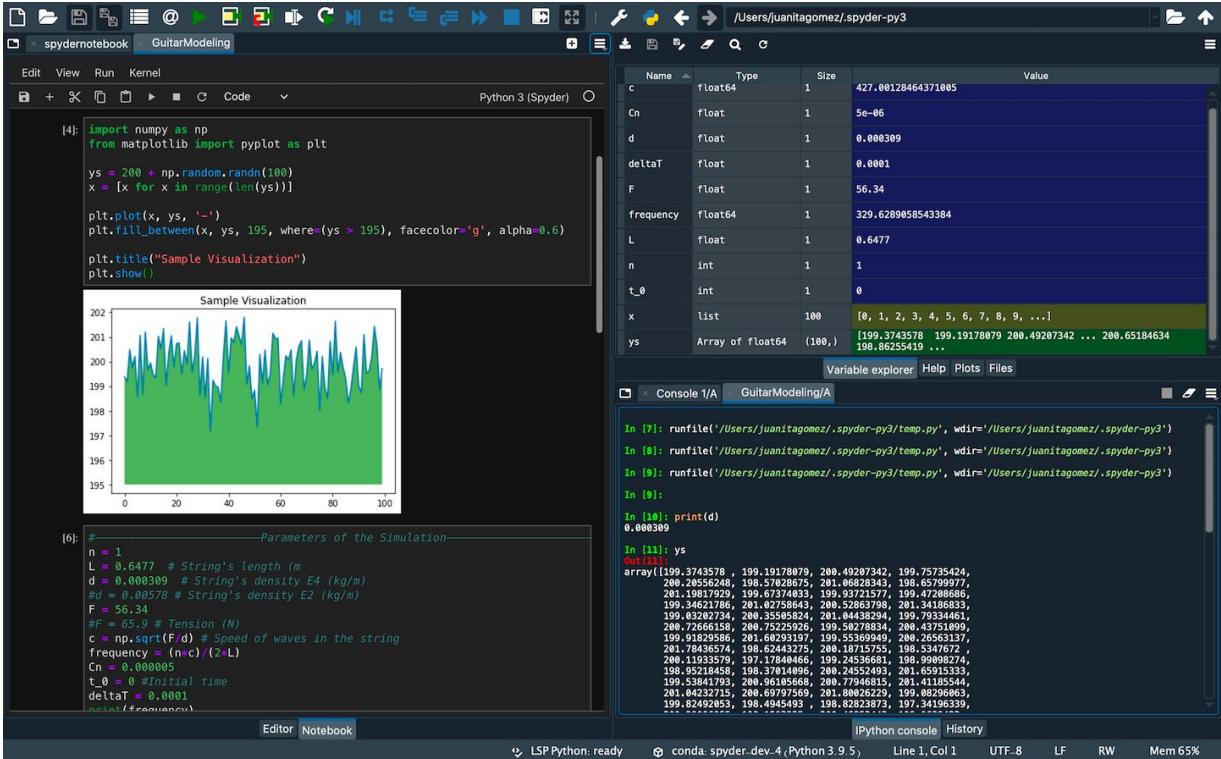


Det finnes mange andre plugins som kan bedre brukeropplevelsen



jupyter_contrib_nbextensions
(channel: conda-forge)

Det finnes en Spyder plug-in for å åpne notebooks



spyder-notebook (channel: spyder-ide)

Jupyter4Edu er en veldig fin ressurs med mye materiale om undervisning med Jupyter notebooks

The screenshot shows a web-based Jupyter notebook interface. On the left, a sidebar displays the table of contents for the book "Teaching and Learning with Jupyter". The chapters listed are:

- 1 Introduction
 - Acknowledgments
- 2 Why we use Jupyter notebooks
 - 2.1 Why do we use Jupyter?
 - 2.2 But first, what is Jupyter Notebo...
 - 2.3 Course benefits & anecdotes
 - 2.4 Student benefits
 - 2.5 Instructor benefits
 - 2.6 Conclusions
- 3 Notebooks in teaching and learning
 - 3.1 Oh the places your notebooks w...
 - 3.2 Before You Begin...
- 4 A catalogue of pedagogical patterns
 - 4.1 Introduction
 - 4.2 Shift-Enter for the win
 - 4.3 Fill in the blanks
 - 4.4 Target Practice

The main content area shows the first chapter, "Teaching and Learning with Jupyter", by a group of authors. The title is "Teaching and Learning with Jupyter". The authors listed are Lorena A. Barba, Lecia J. Barker, Douglas S. Blank, Jed Brown, Allen B. Downey, Timothy George, Lindsey J. Heagy, Kyle T. Mandli, Jason K. Moore, David Lippert, Kyle E. Niemeyer, Ryan R. Watkins, Richard H. West, Elizabeth Wickes, Carol Willing, and Michael Zingale. The date of publication is 2019-12-06.

Chapter 1 Introduction

This handbook is for any educator teaching a topic that includes data analysis or computation in order to support learning. It is not just for educators teaching courses in engineering or science, but also data journalism, business and quantitative economics, data-based decision sciences and policy, quantitative health sciences, and digital humanities. It aims to provide an entry point, and a broad overview of Jupyter in education. Whether you are already using Jupyter to teach, you have found learning materials built on Jupyter that piqued your curiosity, or have never heard of Jupyter, the material in this open book can empower you to use this technology in your teaching.

Project Jupyter is a broad collaboration that develops open-source tools for interactive and exploratory computing. The tools include: over 100 computer languages (with a focus on Python), the Jupyter

<https://jupyter4edu.github.io/jupyter-edu-book/>

Greg Wilson har skrevet en artikkel med 10 tips for kodeundervisning



<https://journals.plos.org/ploscombi/article?id=10.1371/journal.pcbi.1006023>

Kodeskolen har holdt mange kodekurs med ulike programmeringsopplegg

The screenshot shows the GitHub organization page for "Kodeskolen ved Simula". The page displays five public repositories:

- tekna_h21_videre** (Public): Jupyter Notebook, 0 forks, 1 star, 0 issues, 0 pull requests, Updated 4 days ago.
- tekna_innlandet_okt21** (Public): Python, 0 forks, 0 stars, 0 issues, 0 pull requests, Updated 7 days ago. Description: Dette repositoriet er ment for deling av kursmateriale for Introduksjonskurs i Python-programmering, som arrangeres av Tekna og Kodeskolen.
- tekna_h21_intro** (Public): Jupyter Notebook, 0 forks, 0 stars, 0 issues, 0 pull requests, Updated 21 days ago.
- inspiria_h21** (Public): Python, 0 forks, 0 stars, 0 issues, 0 pull requests, Updated on Aug 20.
- tekna_agder_v21_2** (Public): Jupyter Notebook, 0 forks, 0 stars, 0 issues, 0 pull requests, Updated on May 6.

<https://github.com/orgs/kodeskolen/repositories>

Skriv i chatteren: En ting du vil ta med deg videre fra dette kurset

Til sist: Husk evalueringsskjema

