

An aerial photograph of a city, likely Copenhagen, showing a dense urban area with many buildings and streets. A prominent blue line, possibly a river or canal, winds through the city. The image has a greenish tint, suggesting it might be a satellite or aerial view with a specific color filter or overlay.

Python for begyndere — m.strova.dk/python
27. oktober 2026

Mathias Rav
mathias@scalgo.com

SCALGO

Dagens program

- I skal kode i Python for første gang på jeres computer
i Google Colab: <https://colab.research.google.com/>

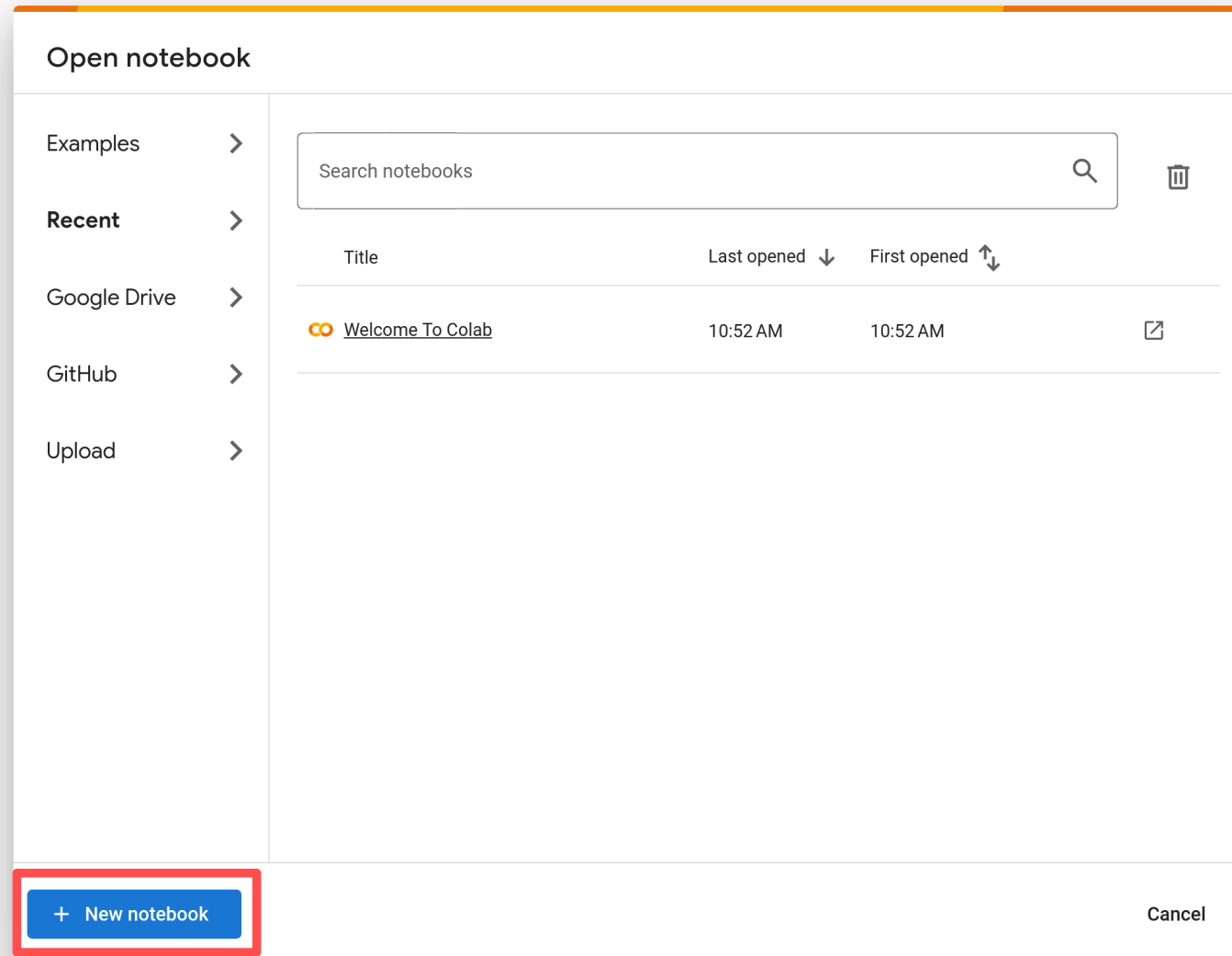
1. Grundlæggende kodning
 - Syntaks
 - Basale pakker
2. Datahåndtering
 - Indlæs data fra csv
 - Plot og håndtér data
 - Gem resultater som csv
3. Numeriske metoder
 - Integration
 - Statistik

Vi kommer knapt nok til at skrabe overfladen,
og foredraget i dag er kun en kickstart.
For at lære Python kræves øvelse, øvelse, øvelse.

Om mig

- Ph.d. i datalogi fra AU (2019)
- Startede med Python da jeg startede på uni (2010)
 - Første store projekt (2012): Tutorhjemmesiden `matfystutor.dk`
- Softwareudvikler hos Scalgo siden 2019
 - hvor vi bruger Python, Rust, C++, TypeScript, ...

colab.research.google.com



Google sign-in required

You must be logged in with a Google Account to continue.

Cancel

[Sign in](#)

colab.research.google.com

The image shows the Google Colab web interface. On the left, the 'Open notebook' sidebar is visible, featuring a search bar and a list of options: Examples, Recent, Google Drive, GitHub, and Upload. A red box highlights the '+ New notebook' button at the bottom of this sidebar, with a red arrow pointing from it to the main editor area. The main editor area displays a new notebook titled 'Untitled0.ipynb'. The top menu bar includes 'File', 'Edit', 'View', 'Insert', 'Runtime', 'Tools', and 'Help', with a status message 'All changes saved'. Below the menu, there are tabs for '+ Code' and '+ Text'. The central workspace contains a prompt: 'Start coding or generate with AI.' with a play button icon. On the right side, there is a 'Release notes' panel for the date '2024-09-23'. It contains a list of updates and a section for 'Python package upgrades' with a list of package versions.

Open notebook

- Examples >
- Recent >
- Google Drive >
- GitHub >
- Upload >

+ New notebook

Untitled0.ipynb ☆

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

Start coding or generate with AI.

Release notes ✕

Please follow our [blog](#) to see more information about new features, tips and tricks, and featured notebooks such as [Analyzing a Bank Failure with Colab](#).

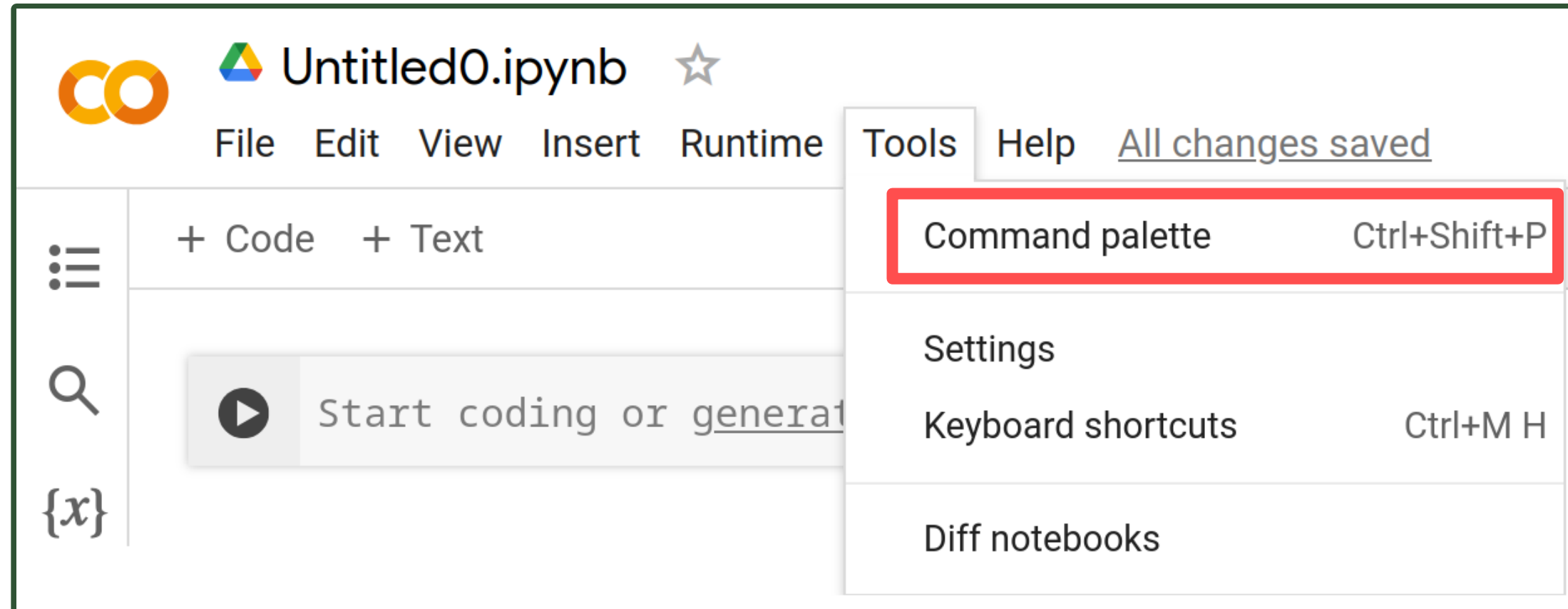
2024-09-23

- Improved code snippet search
- Updated Marketplace image and public local runtime container
- Improved the look-and-feel of interactive form dropdowns and checkboxes
- Fixed bugs
 - activating the skip link caused the notebook to scroll out of view
 - toggling a checkbox too much caused the page to crash
 - lightning fast drags could cause orphaned tabs
 - custom widgets snippet would show for local runtimes

Python package upgrades

- accelerate 0.32.1 -> 0.34.2
- arviz 0.18.0 -> 0.19
- autograd 1.6.2 -> 1.7.0
- bigframes 1.14.0 -> 1.18.0
- dask 2024.7.1 -> 2024.8.0
- distributed 2024.7.1 -> 2024.8.0
- duckdb 0.10.3 -> 1.1.0
- earthengine-api 0.1.416 -> 1.0.0
- flax 0.8.4 -> 0.8.5
- gdown 5.1.0 -> 5.2.0
- geemap 0.33.1 -> 0.34.3
- geopandas 0.14.4 -> 1.0.1
- google-cloud-aiplatform 1.59.0 -> 1.67.1

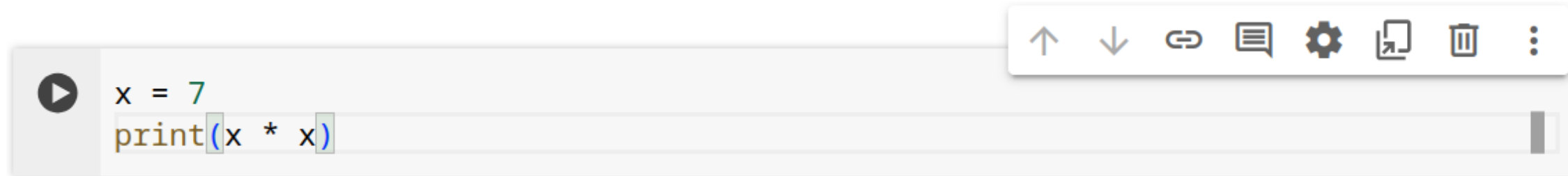
colab.research.google.com



Nyttige commands: Insert "Above" og Insert "Below" – indsætter nye celler
— "Delete", "Split" og "Merge" – sletter/deler/samler celler

Firefox-brugere: Gå ind i Keyboard shortcuts og ændr Command palette shortcut

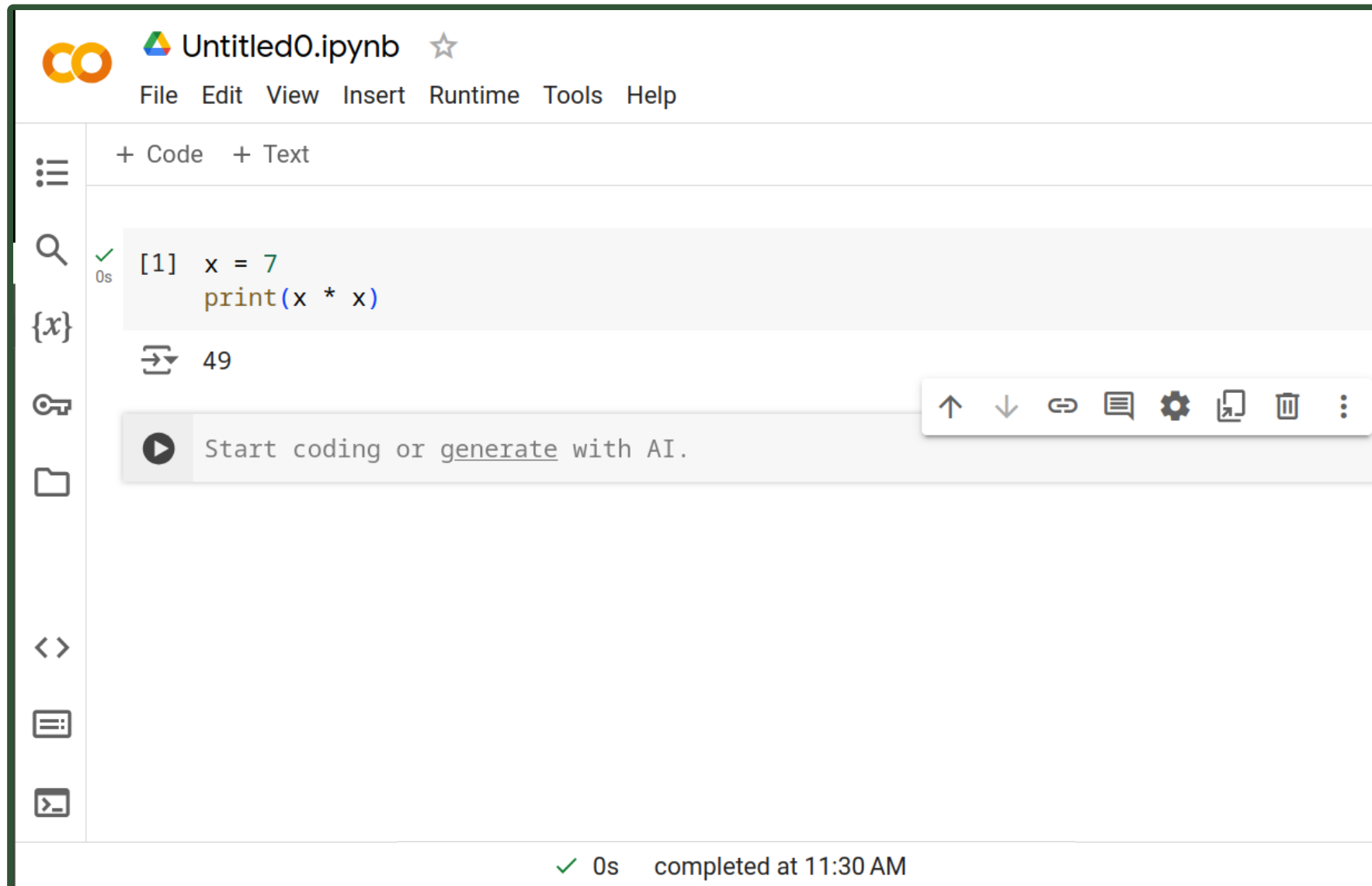
colab.research.google.com

A screenshot of the Google Colab code editor interface. It shows a code cell with a play button icon on the left. The code inside the cell is `x = 7` followed by `print(x * x)` on the next line. The cursor is positioned at the end of the second line. To the right of the code cell is a toolbar with icons for undo, redo, insert link, insert comment, settings, copy, delete, and a menu icon.

```
x = 7
print(x * x)
```

Tryk Shift-Enter

colab.research.google.com



Colab er ikke gratis!

- Google stiller en gratis udgave til rådighed fordi de gerne vil have jer til at bruge betalingsudgaven...
- Sørg for at få installeret Python og Jupyter på jeres egen computer!
— men for at komme hurtigt i gang, så kan I nøjes med Colab under foredraget i dag.

Runtime disconnected

Your runtime has been disconnected due to inactivity or reaching its maximum duration. [Learn more](#)

If you are interested in longer runtimes with more lenient timeouts, you may want to check out [Colab Pro](#).

Close

[Reconnect](#)

```
x = 3
```

```
y = 5
```

```
x = 2
```

```
print(x * y)
```

~~x = 3~~

y = 5

x = 2

Senere tildeling overskriver tidligere værdi af x

print(x * y)

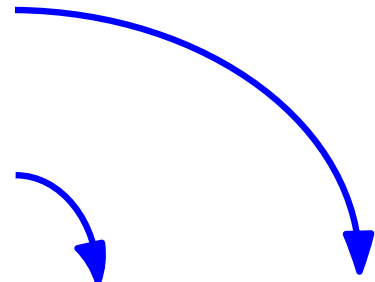
x = 3

y = 5

x = 2

print(x * y)

2 * 5 = 10



```
x = 4
```

```
y = x + 1
```

```
z = 2 * y
```

```
print(z)
```

```
x = 4
```

```
x = x + 1
```

```
x = 2 * x
```

```
print(x)
```

```
x = 4
```

```
y = x + 1
```

```
z = 2 * y
```

```
print(z)
```

Hvilket program er nemmere at forstå?

```
a = [3, 5, 7]
print(a)
print(a[0])
print(a[1] + a[2])
print(type(a))
print(len(a))
```

```
a = [3, 5, 7]
```

```
for e in a:  
    print(e)
```

```
a = [3, 5, 7]
```

```
sum(a)
```

```
min(a)
```

```
max(a)
```

```
5 in a
```

```
6 in a
```

```
7 not in a
```

```
a.index(7)
```

```
a.count(6)
```

```
import numpy as np

b = np.array([3, 5, 7])
print(b)
print(b[0])
print(b[1] + b[2])
print(type(b))
print(b.dtype)
print(len(b))
print(b.shape)
```

```
import numpy as np
```

Python list:

```
sum(a)  
min(a)  
max(a)  
5 in a  
6 in a  
7 not in a  
a.index(7)  
a.count(3)
```

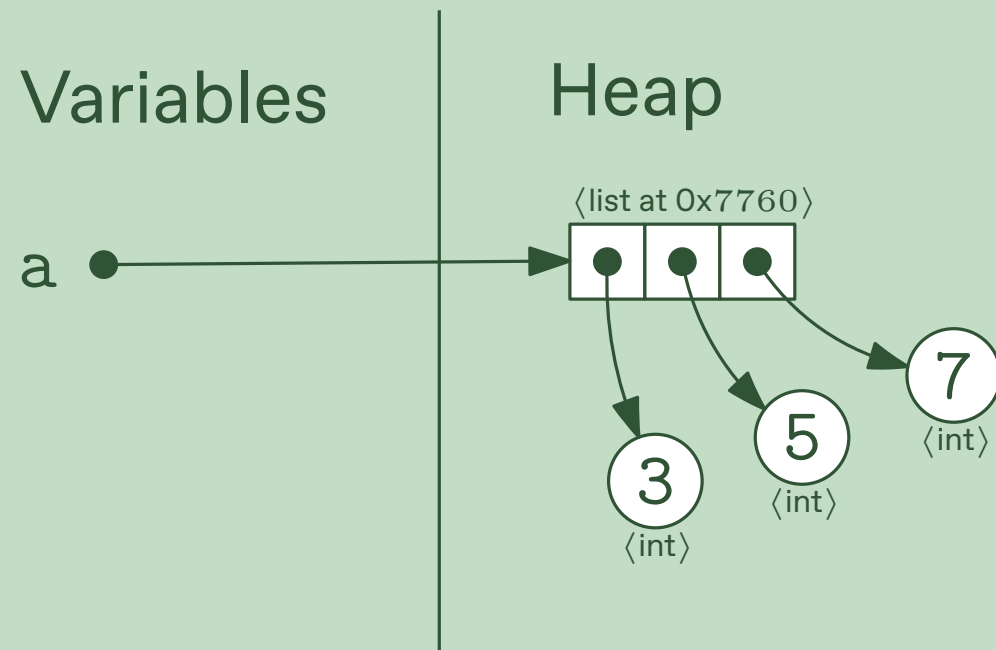
NumPy array:

```
np.sum(b)  
np.min(b)  
np.max(b)  
5 in b  
6 in b  
7 not in b  
(b == 7).nonzero()[0]  
(b == 3).sum()
```

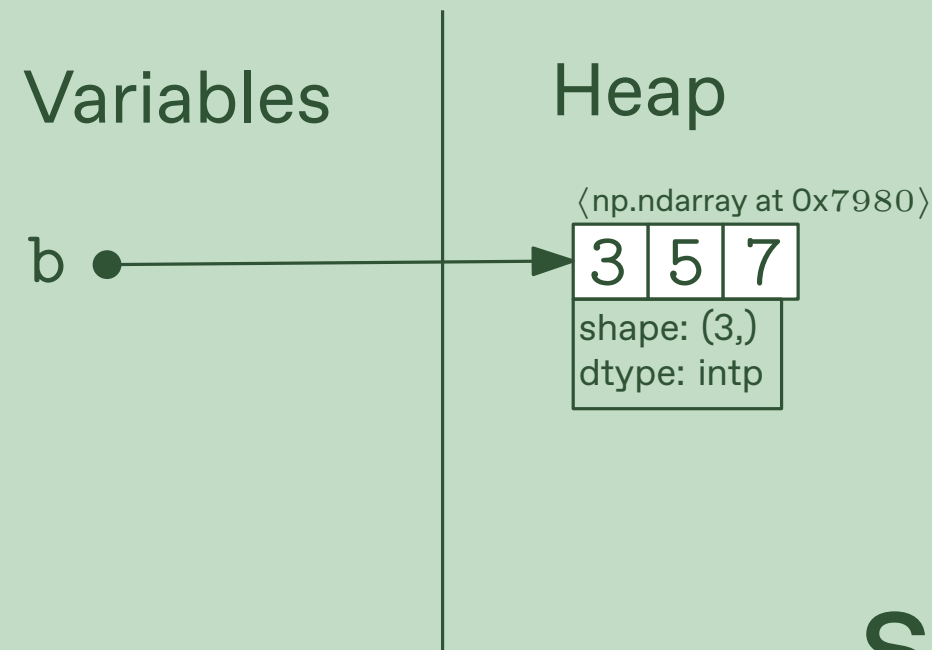
Standard library vs. external library

- Python har en række indbyggede ting, såsom lister `a = [3, 5, 7]`
- Men mange ting kræver eksterne pakker, f.eks. `import numpy as np`

`a = [3, 5, 7]`



`b = np.array([3, 5, 7])`



Standard library vs. external library

```
a = [3, 5, 7]
```

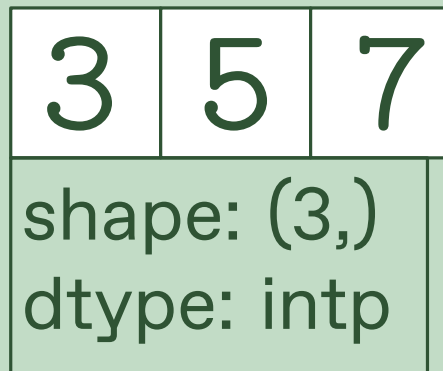
⟨list at 0x7760⟩



```
import numpy as np
```

```
b = np.array([3, 5, 7])
```

⟨np.ndarray at 0x7980⟩



np.array kan kun
indeholde tal

- Andre pakker: matplotlib, pandas, scipy, ...

Standard library vs. external library

```
a = [3, 5, 7]
a2 = a * 2
print(a2)
```

→ [3, 5, 7, 3, 5, 7]

```
print(a == 5)
```

→ False

”Nej, en liste er ikke
lig med et tal”

```
import numpy as np
b = np.array([3, 5, 7])
b2 = b * 2
print(b2)
```

→ [6 10 14]

```
print(b == 5)
```

→ [False True False]

”Tjo, nogle af tallene
er lig med 5”

Er I forvirrede?

- Hvis I ikke er forvirrede nu, så bliver I det senere
- Vigtige forskelle på standard lister og NumPy arrays
 - Vi kaster os ud i NumPy med det samme
 - Andre Python-kurser starter med ”ren standard Python”



Introduction to Programming with Scientific Applications

Aarhus University, Department of Computer Science

Welcome

Course plan

- Compact
- All slides (pdf)
- with answers (pdf)

Exercises

Handins

Final project

- I - Medians
- II - Portfolio
- III - NMR
- IV - MNIST
- V - Open topic

Exam

- Statistics
- Past exams

Plagiarism & GAI

Workload

Literature

Python installation

Windows 11

Welcome

This page contains the public content for the course *Introduction to Programming with Scientific Applications* offered by Aarhus University, Department of Computer Science. The lecturer is [Gerth Stølting Brodal](#).

The courses is listed in the Aarhus University course catalogue with the two entries:

- [Introduction to Programming with Scientific Applications \(DK\)](#)
- [Introduction to Programming with Scientific Applications \(EN\)](#)

Class lists, discussions, student assignment submissions and feedback are handled using Aarhus University's learning management system Brightspace at brightspace.au.dk.

The course gives an introduction to the Python 3 programming language and applications using Python. Throughout the course students are encouraged to seek online information in, e.g., the Python language specification.

The course will be run with weekly 2 x 2 hours lectures (alternatively recorded lectures on YouTube), 3 hours of



Tyvstart på:
gsbrodal.github.io/ipsa/

— Andre Python-kurser starter med ”ren standard Python”

Glæd jer til IPSA! (”Python med Gerth”)

SCALGO

Let's create space for water

Course overview

Basic programming
Advanced / specific python
Libraries & applications

1. Introduction to Python	10. Functions as objects	19. Linear programming
2. Python basics / if	11. Object oriented programming	20. Generators, iterators, with
3. Basic operations	12. Class hierarchies	21. Modules and packages
4. Lists / while / for	13. Exceptions and files	22. Working with text
5. Tuples / comprehensions	14. Doc, testing, debugging	23. Relational data
6. Dictionaries and sets	15. Decorators	24. Clustering
7. Functions	16. Dynamic programming	25. Graphical user interfaces (GUI)
8. Recursion	17. Visualization and optimization	26. Java vs Python
9. Recursion and Iteration	18. Multi-dimensional data	27. Final lecture

10 handins
1 final project (last 1 month)

Tyvstart på:
gsbrodal.github.io/ipsa/

— Andre Python-kurser starter med ”ren standard Python”

Glæd jer til IPSA! (”Python med Gerth”)

Strings & slicing

- En streng er en følge af bogstaver (eller talcifre, symboler, mellemrum, ...)
`s = 'abwwdexy___lwtavghevt_xypxxyattx_hxwoadnxxx'`
- Det ligner jo volapyk...
- `print(s[2:42:3])`

✓
0s



```
s = 'abwwdexy___lwtavghevt_xypxxyattx_hxwoadnxxx'  
print(s[2:42:3])
```



```
we_love_python
```

Strings & slicing

- En streng er en følge af bogstaver (eller talcifre, symboler, mellemrum, ...)

```
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44
s = 'abwwdexy___lwttopavghevt_xypxxyattx_hxwoadnxxx'
```

- Det ligner jo volapyk...
- `print(s[2:42:3])`

✓
0s



```
s = 'abwwdexy___lwttopavghevt_xypxxyattx_hxwoadnxxx'
print(s[2:42:3])
```



```
we_love_python
```

Strings & slicing

- En streng er en følge af bogstaver (eller talcifre, symboler, mellemrum, ...)

```
s = 'abwwdexy___lwtopavghevt_xypxxyattx_hxwoadnxxx'
```

The string is displayed with indices 0 to 44 above each character. The characters from index 2 to 42 are highlighted with a blue box, and the step value 3 in the slicing operation is highlighted with a red box.

- Det ligner jo volapyk...
- `print(s[2:42:3])`

✓
0s



```
s = 'abwwdexy___lwtopavghevt_xypxxyattx_hxwoadnxxx'  
print(s[2:42:3])
```



```
we_love_python
```

Slicing

- `s[2:42:3]`: Fra position 2 lige indtil position 42, tag hvert tredje element
- `s[2:42]`: Fra position 2 lige indtil position 42, alle elementer
- `s[2:]`: Fra position 2 til slutningen
- `s[:42]`: Fra starten lige indtil position 42
- `s[: :3]`: Fra start til slut, tag hvert tredje element
- `s[:]`: Fra start til slut, dvs. alting

Slicing

- `s[2:42:3]`: Fra position 2 lige indtil position 42, tag hvert tredje element
- `s[2:42]`: Fra position 2 lige indtil position 42, alle elementer

Bemærk! Halvåbne intervaller!

2 er inkluderet, 42 er IKKE inkluderet

Slicing: lists vs. numpy

```
aold = [13, 27, 7, 42]
```

```
anew = aold
```

```
anew[2] = 12
```

```
print(f"aold is {aold}")
```

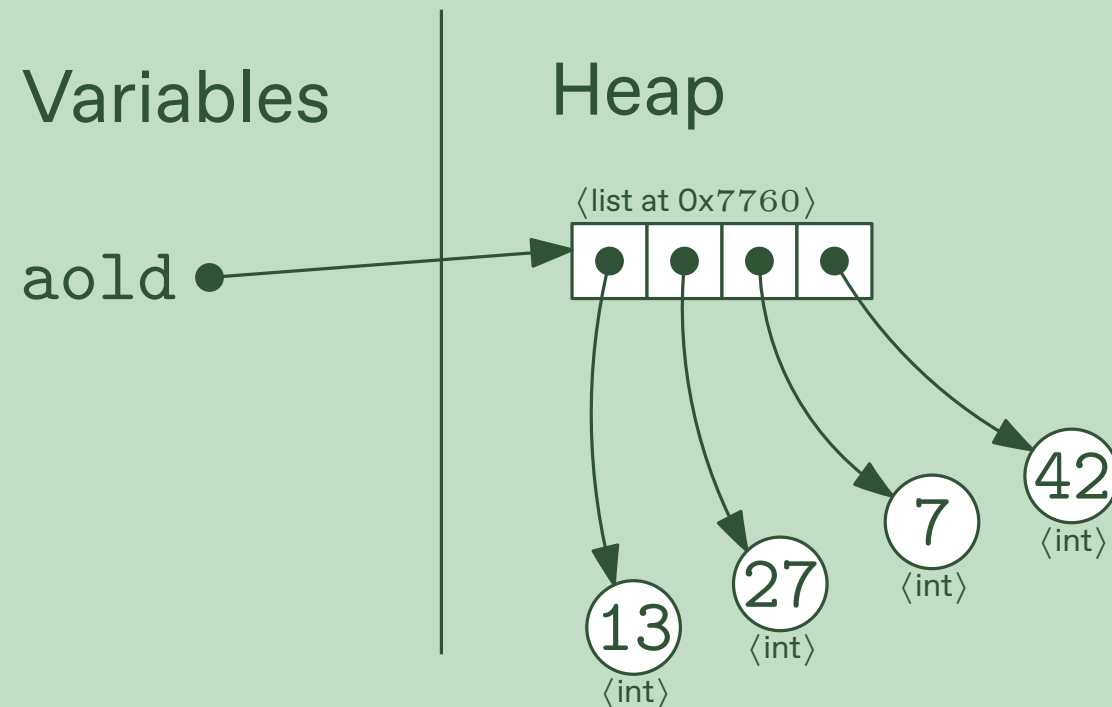
```
print(f"anew is {anew}")
```



f-strings indsætter værdien af aold
der hvor der står {aold}

Slicing: lists vs. numpy

```
→ aold = [13, 27, 7, 42]  
   anew = aold  
   anew[2] = 12  
   print(f"aold is {aold}")  
   print(f"anew is {anew}")
```



Slicing: lists vs. numpy

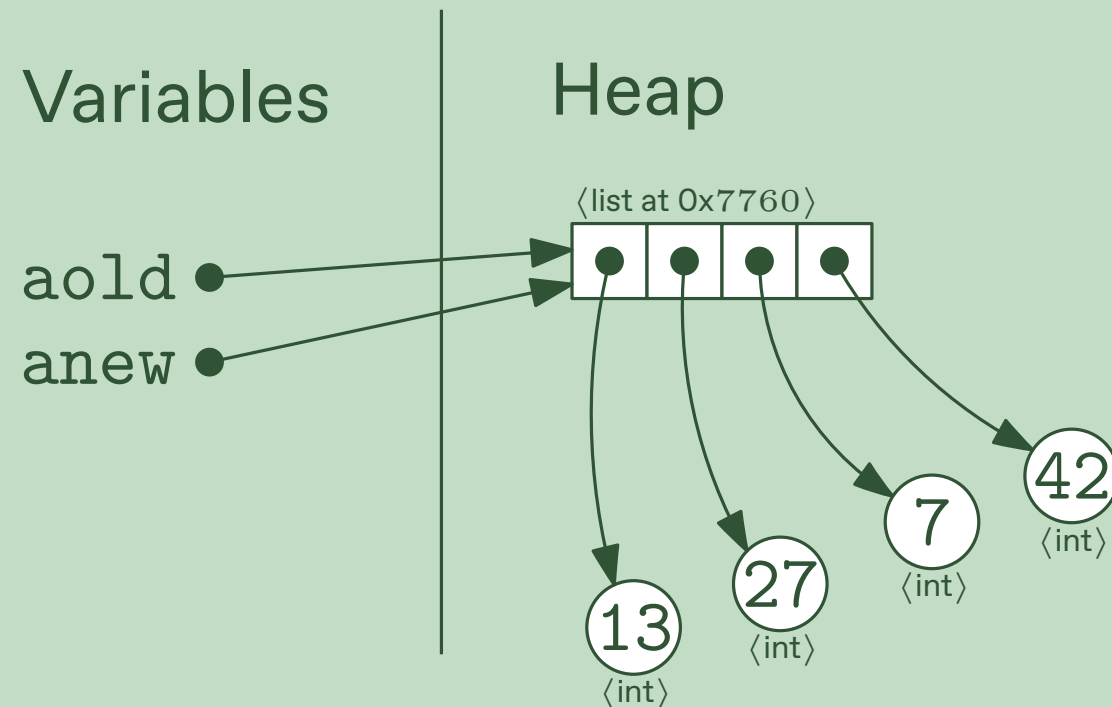
```
aold = [13, 27, 7, 42]
```

```
→ anew = aold
```

```
anew[2] = 12
```

```
print(f"aold is {aold}")
```

```
print(f"anew is {anew}")
```



Slicing: lists vs. numpy

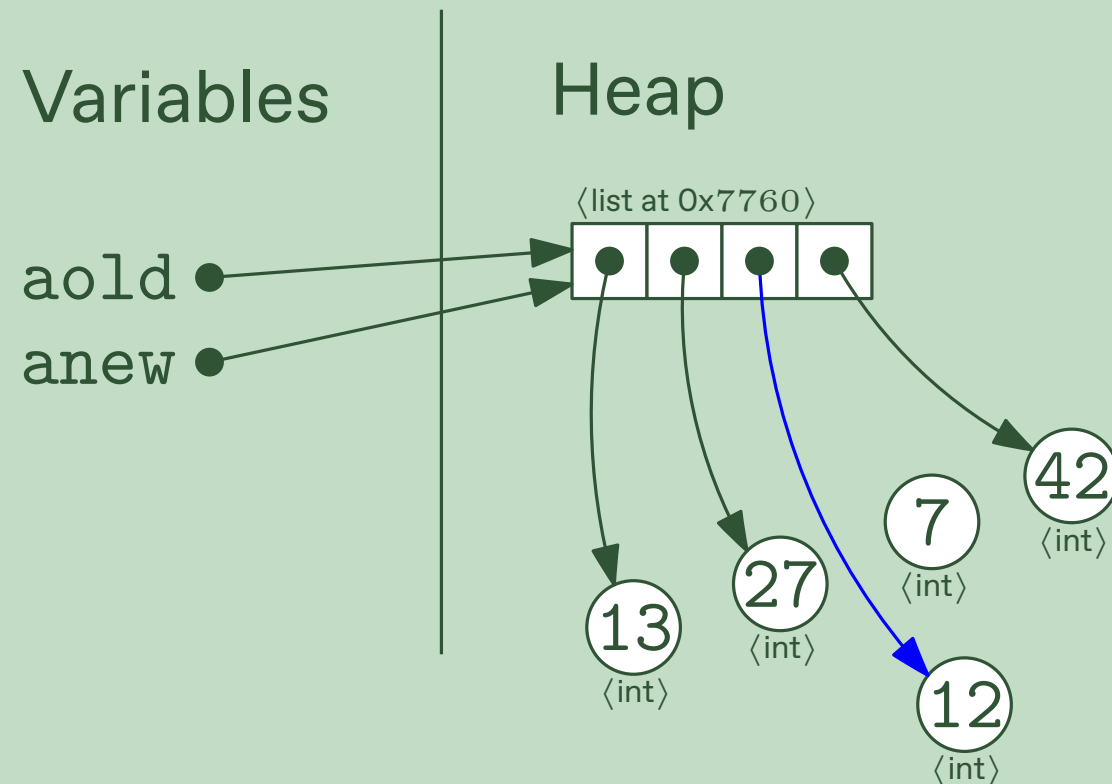
```
aold = [13, 27, 7, 42]
```

```
anew = aold
```

```
→ anew[2] = 12
```

```
print(f"aold is {aold}")
```

```
print(f"anew is {anew}")
```



Slicing: lists vs. numpy

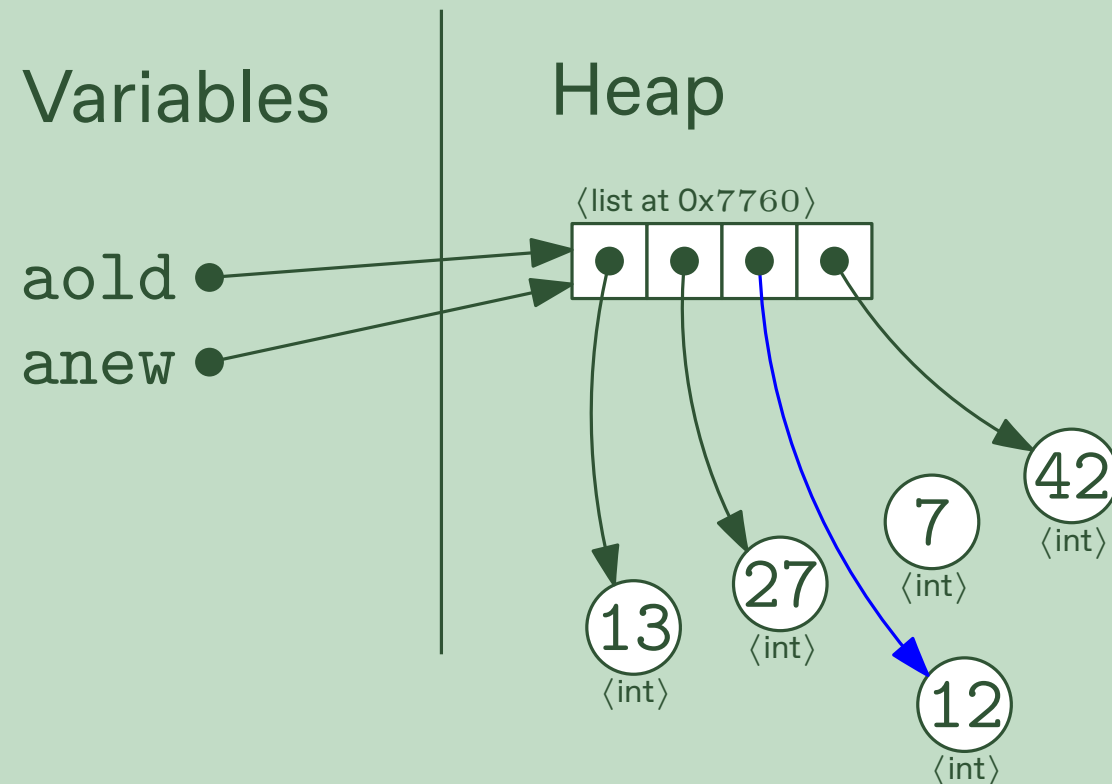
```
aold = [13, 27, 7, 42]
```

```
anew = aold
```

```
anew[2] = 12
```

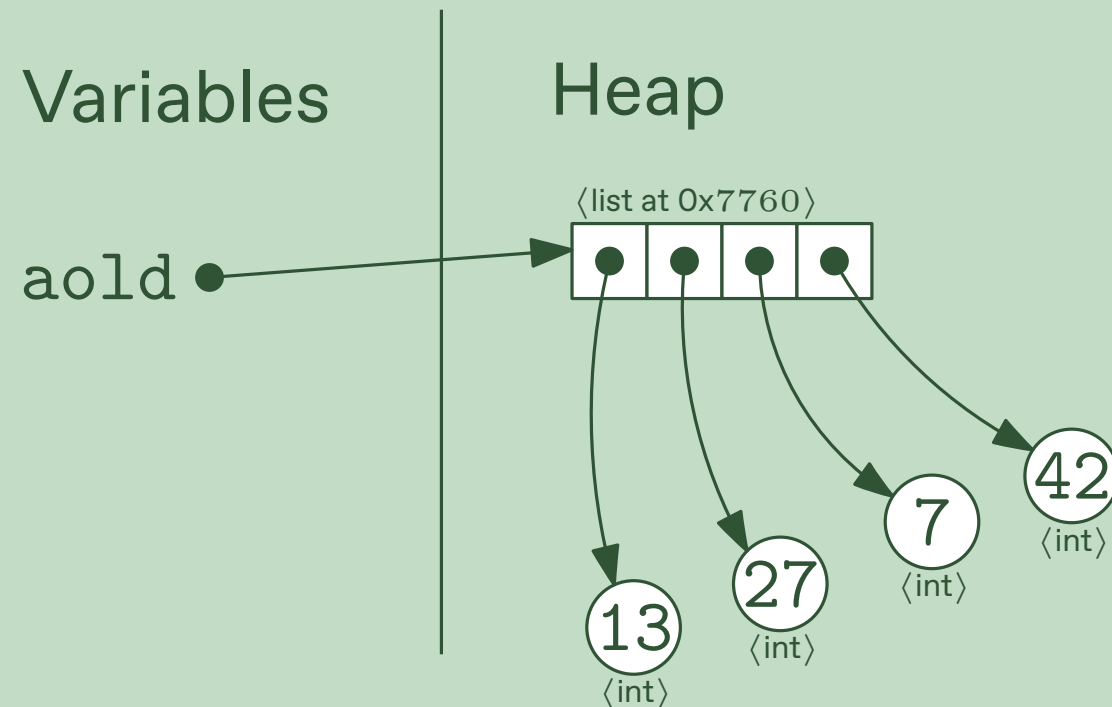
```
print(f"aold is {aold}") aold is [13, 27, 12, 42]
```

```
→ print(f"anew is {anew}") anew is [13, 27, 12, 42]
```



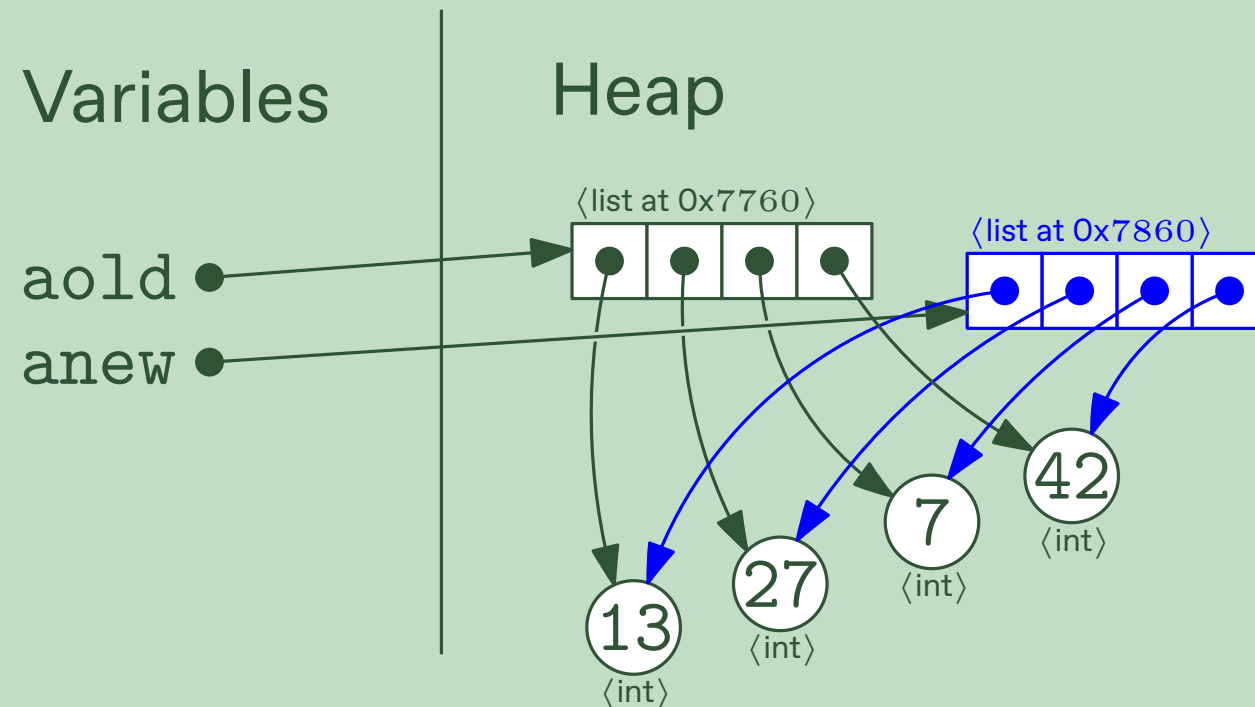
Slicing: lists vs. numpy

```
→ aold = [13, 27, 7, 42]  
   anew = aold[:]  
   anew[2] = 12  
   print(f"aold is {aold}")  
   print(f"anew is {anew}")
```



Slicing: lists vs. numpy

```
aold = [13, 27, 7, 42]
→ anew = aold[:2]
anew[2] = 12
print(f"aold is {aold}")
print(f"anew is {anew}")
```



Slicing: lists vs. numpy

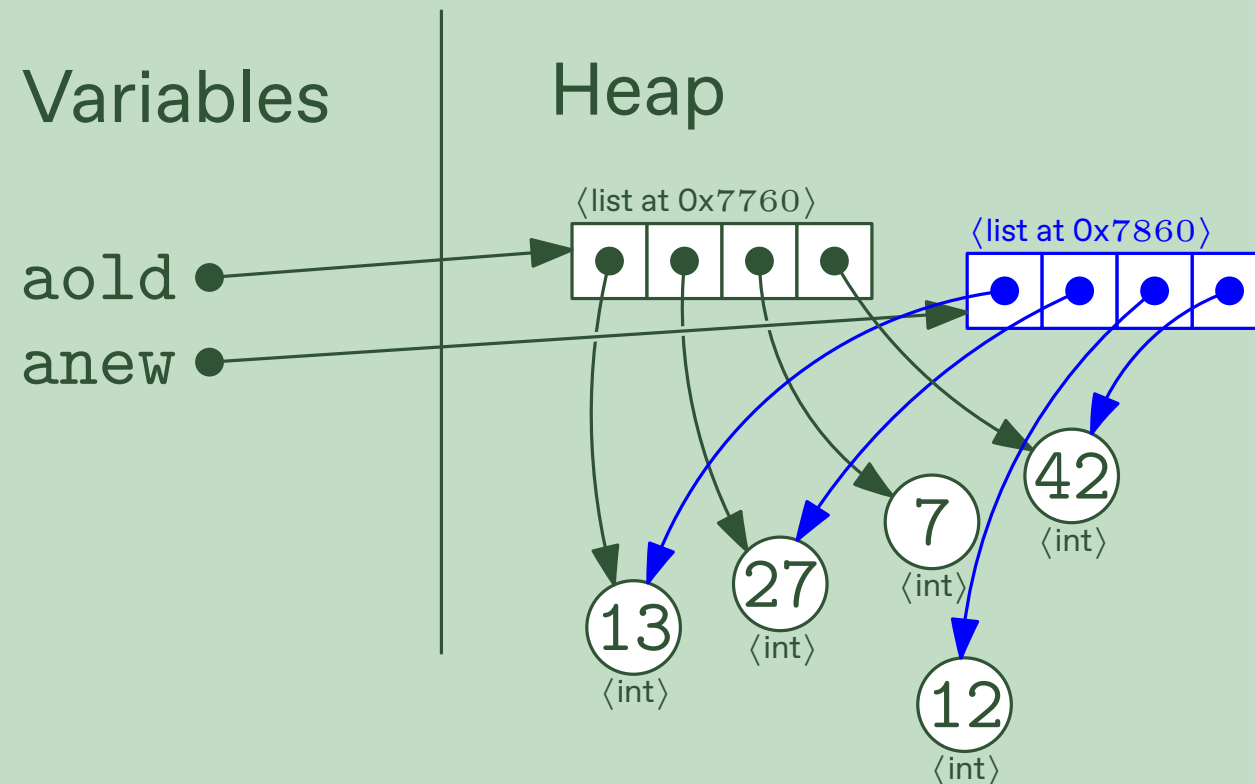
```
aold = [13, 27, 7, 42]
```

```
anew = aold[:]
```

```
→ anew[2] = 12
```

```
print(f"aold is {aold}")
```

```
print(f"anew is {anew}")
```



Slicing: lists vs. numpy

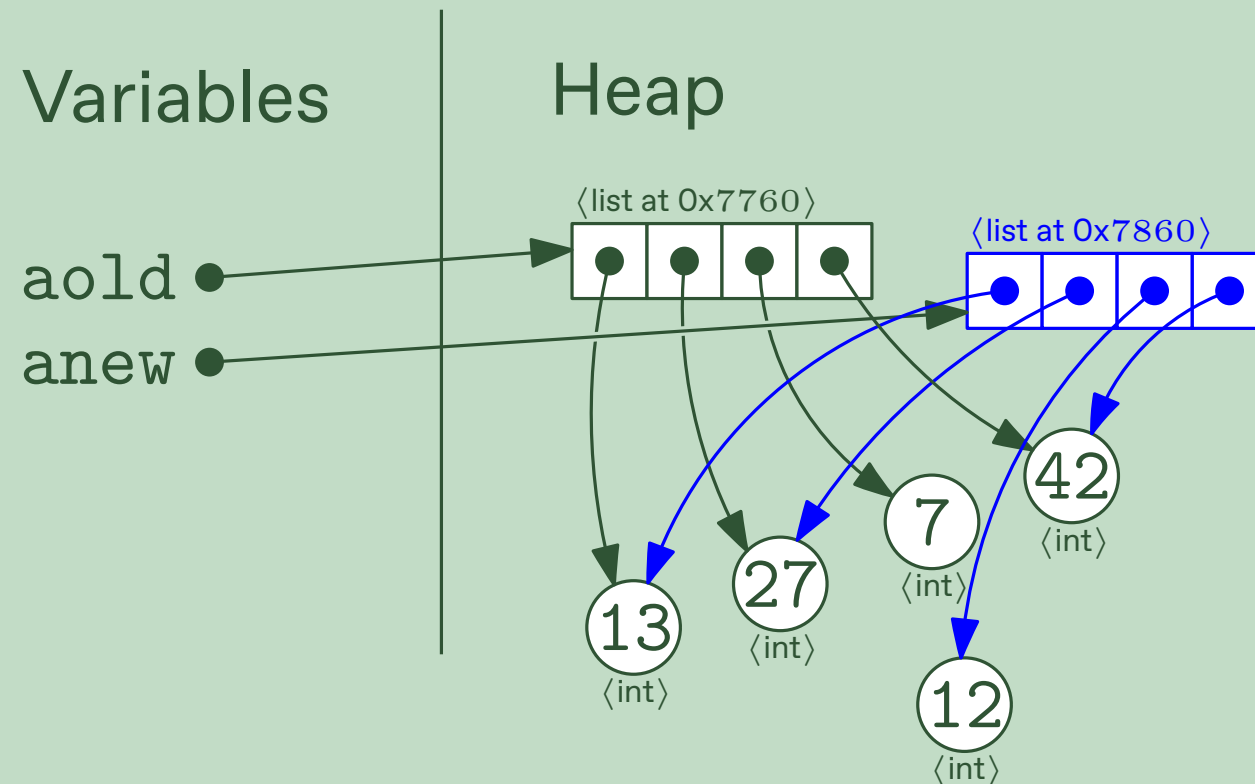
```
aold = [13, 27, 7, 42]
```

```
anew = aold[:]
```

```
anew[2] = 12
```

```
print(f"aold is {aold}") aold is [13, 27, 7, 42]
```

```
→ print(f"anew is {anew}") anew is [13, 27, 12, 42]
```

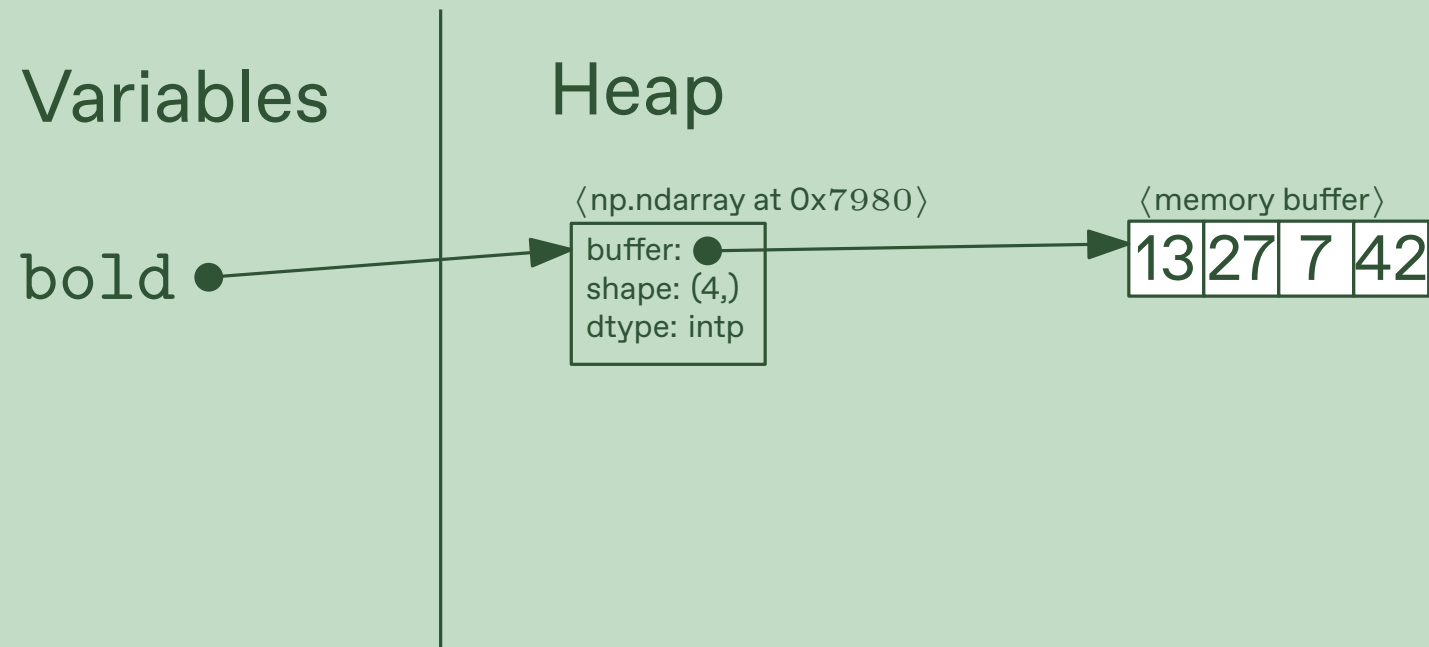
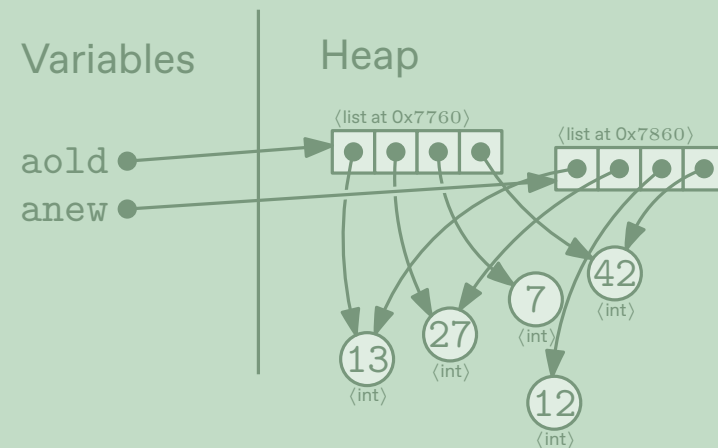


```
import numpy as np
```

Slicing: lists vs. numpy

```
aold = [13, 27, 7, 42]
anew = aold[:]
anew[2] = 12
print(f"aold is {aold}")
print(f"anew is {anew}")
```

```
→ bold = np.array([13, 27, 7, 42])
bnew = bold[:]
bnew[2] = 12
print(f"bold is {bold}")
print(f"bnew is {bnew}")
```

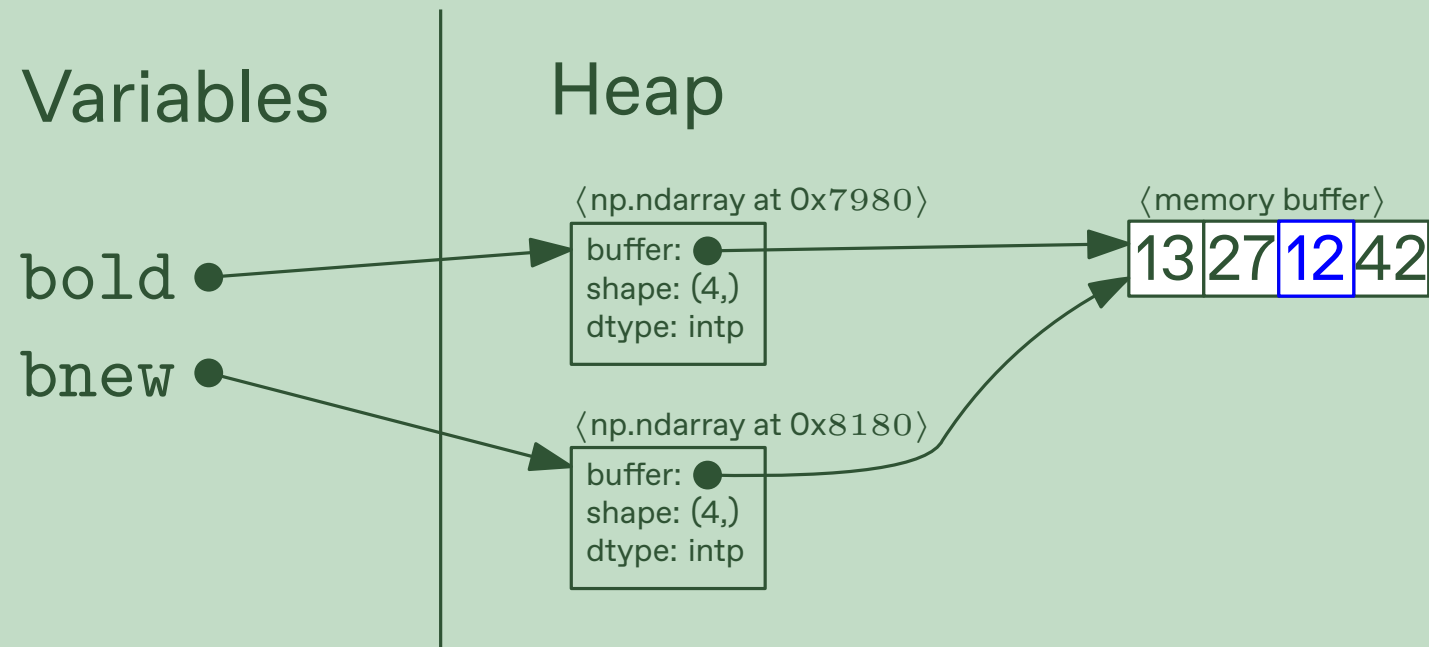
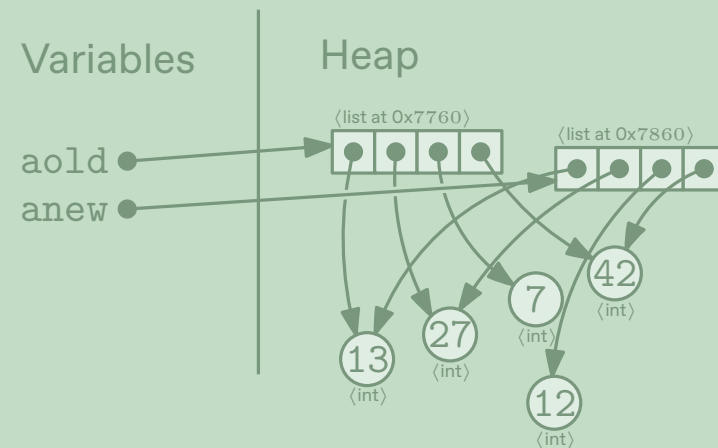


```
import numpy as np
```

Slicing: lists vs. numpy

```
aold = [13, 27, 7, 42]
anew = aold[:]
anew[2] = 12
print(f"aold is {aold}")
print(f"anew is {anew}")
```

```
bold = np.array([13, 27, 7, 42])
bnew = bold[:]
→ bnew[2] = 12
print(f"bold is {bold}")
print(f"bnew is {bnew}")
```



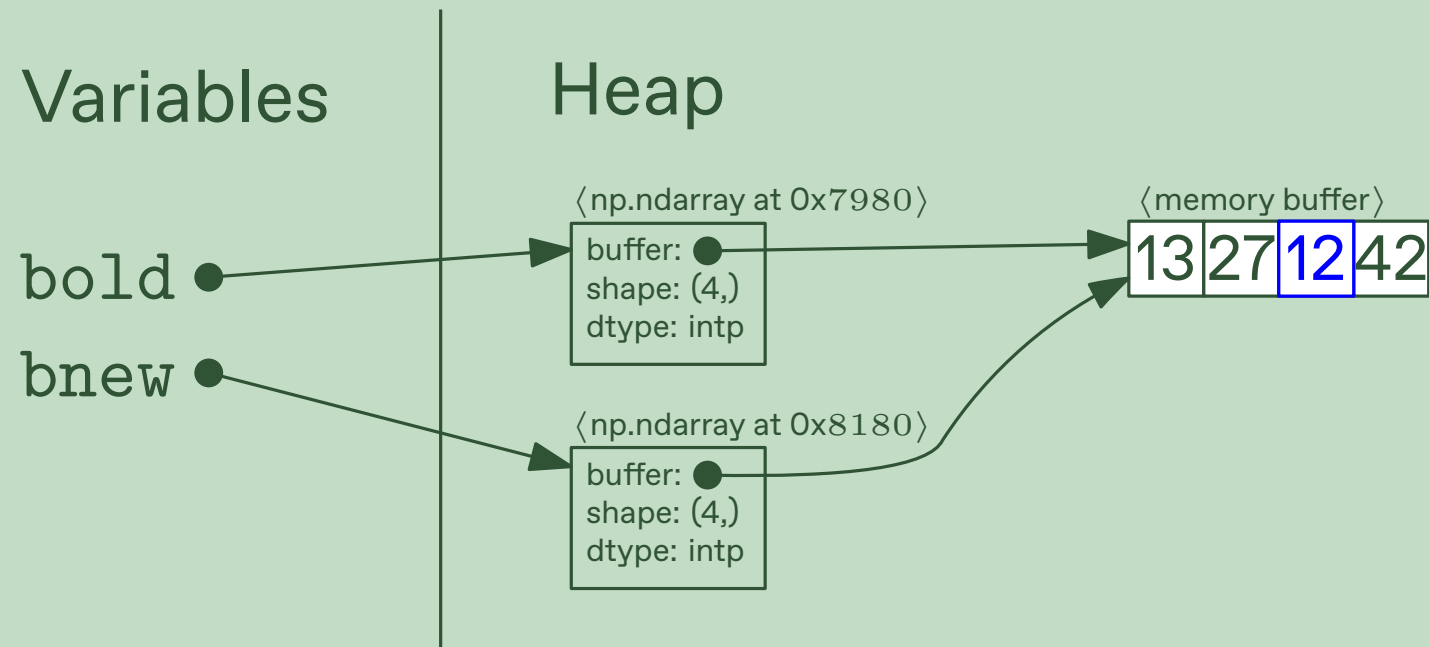
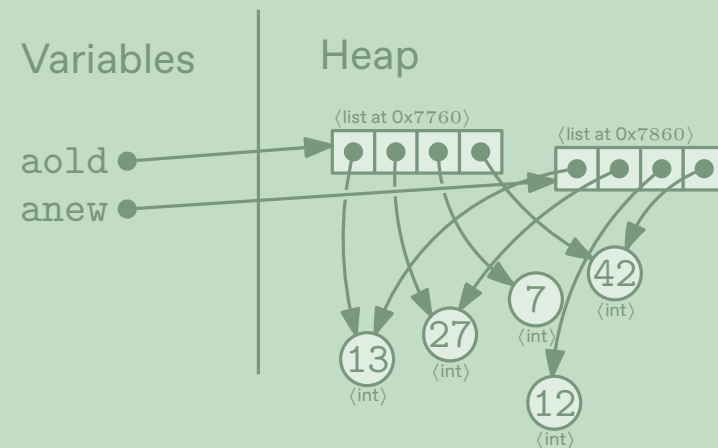
```
import numpy as np
```

Slicing: lists vs. numpy

```
aold = [13, 27, 7, 42]
anew = aold[:]
anew[2] = 12
print(f"aold is {aold}")
print(f"anew is {anew}")
```

```
bold = np.array([13, 27, 7, 42])
bnew = bold[:]
bnew[2] = 12
print(f"bold is {bold}")
→ print(f"bnew is {bnew}")
```

[13 27 12 42]
[13 27 12 42]

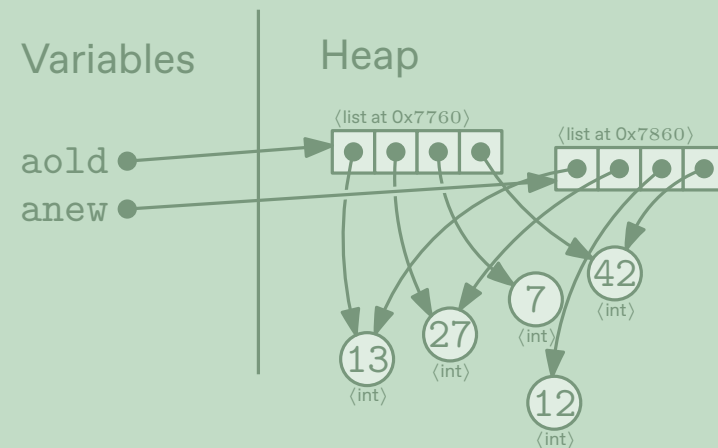


```
import numpy as np
```

Slicing: lists vs. numpy

```
aold = [13, 27, 7, 42]
anew = aold[:]
anew[2] = 12
print(f"aold is {aold}")
print(f"anew is {anew}")
```

```
bold = np.array([13, 27, 7, 42])
bnew = bold.copy()
bnew[2] = 12
print(f"bold is {bold}")
print(f"bnew is {bnew}")
```



Variables

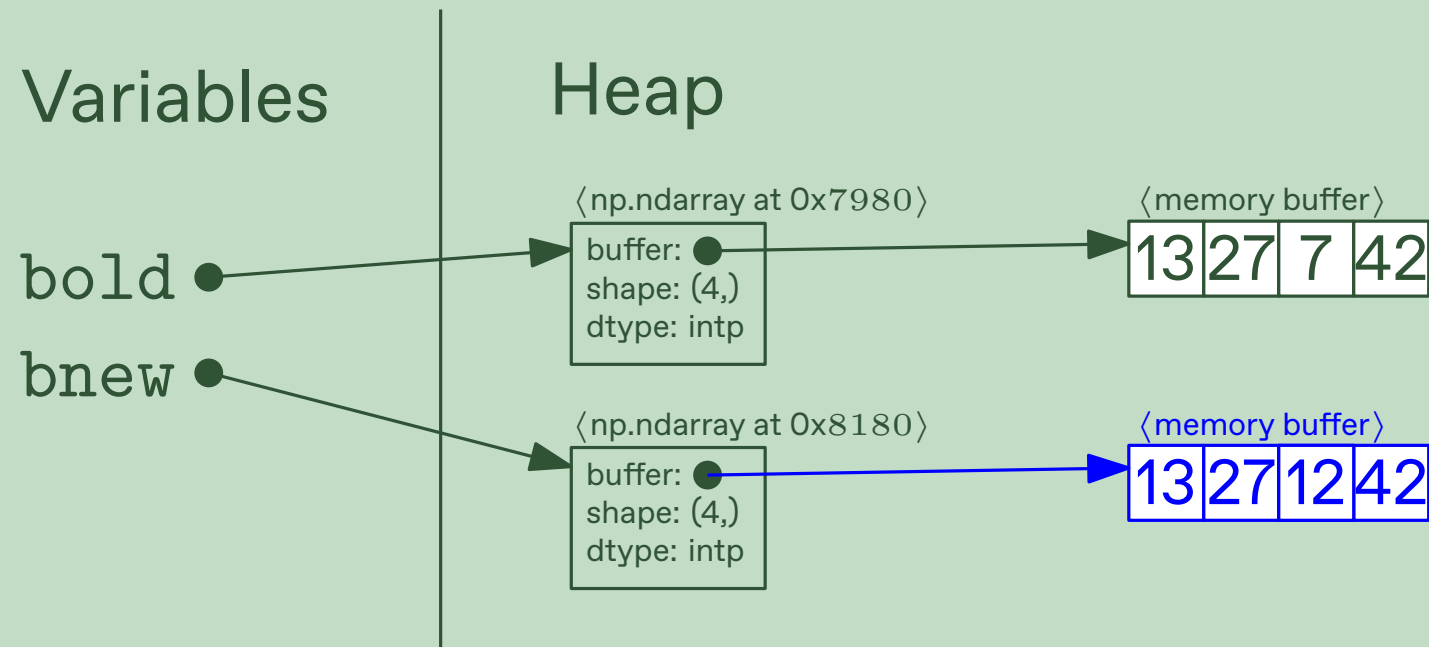
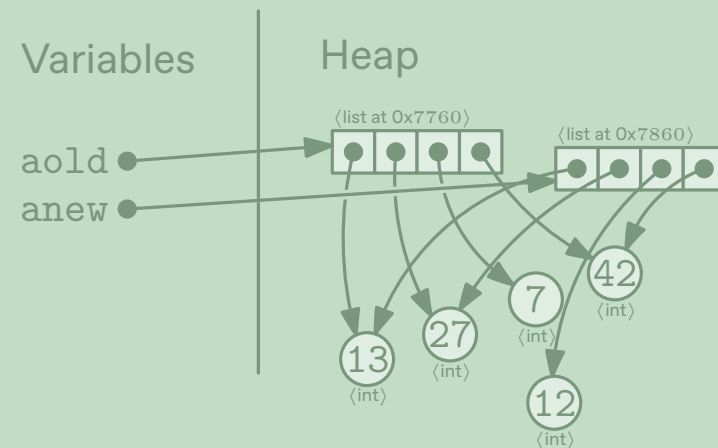
Heap

```
import numpy as np
```

Slicing: lists vs. numpy

```
aold = [13, 27, 7, 42]
anew = aold[:]
anew[2] = 12
print(f"aold is {aold}")
print(f"anew is {anew}")
```

```
bold = np.array([13, 27, 7, 42])
bnew = bold.copy()
→ bnew[2] = 12
print(f"bold is {bold}")
print(f"bnew is {bnew}")
```

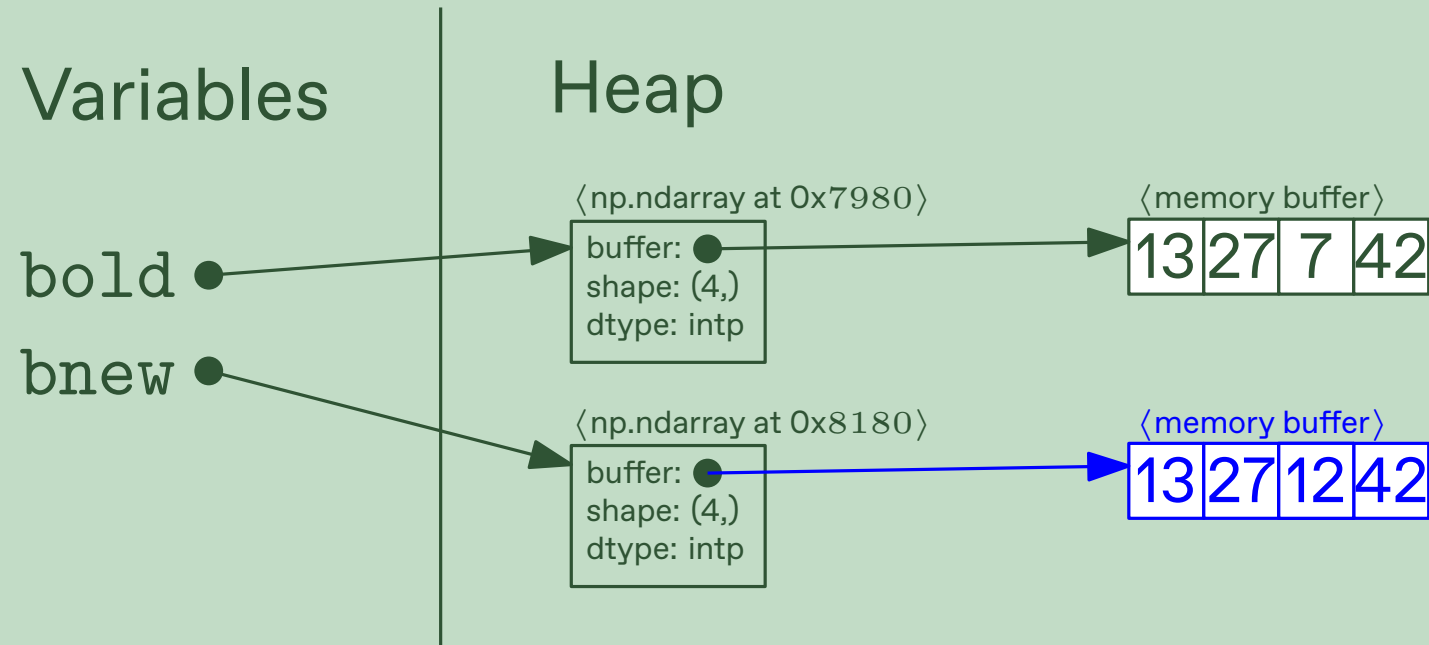
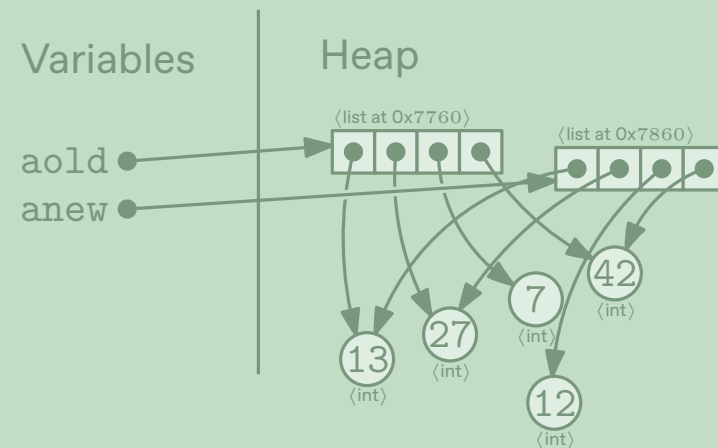


```
import numpy as np
```

Slicing: lists vs. numpy

```
aold = [13, 27, 7, 42]
anew = aold[:]
anew[2] = 12
print(f"aold is {aold}")
print(f"anew is {anew}")
```

```
bold = np.array([13, 27, 7, 42])
bnew = bold.copy()
bnew[2] = 12
print(f"bold is {bold}")    [13 27 7 42]
→ print(f"bnew is {bnew}") [13 27 12 42]
```



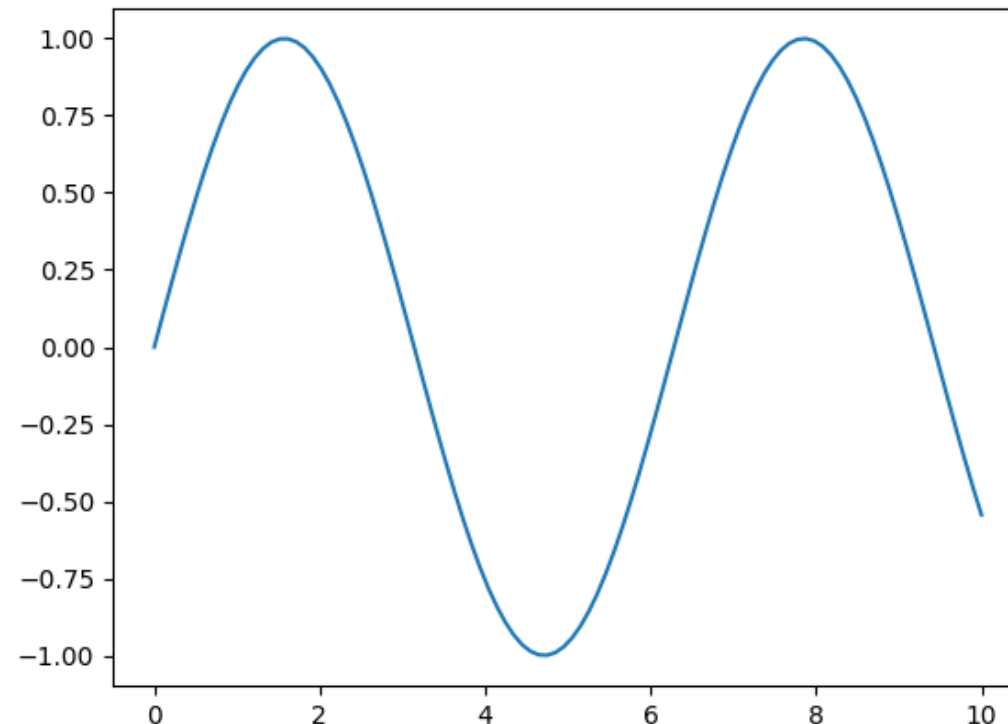
Slicing: lists vs. numpy

- `a[:]`, `a[2:42:3]`, osv. kaldes slicing
- For standard lister: slicing kopierer data
- For numpy arrays: slicing laver et view af den samme buffer
 - Brug `.copy()` for at lave en kopi af data

matplotlib

```
import numpy as np
import matplotlib.pyplot as plt
x = np.linspace(0, 10, 100)
y = np.sin(x)
plt.plot(x, y)
plt.show()
```

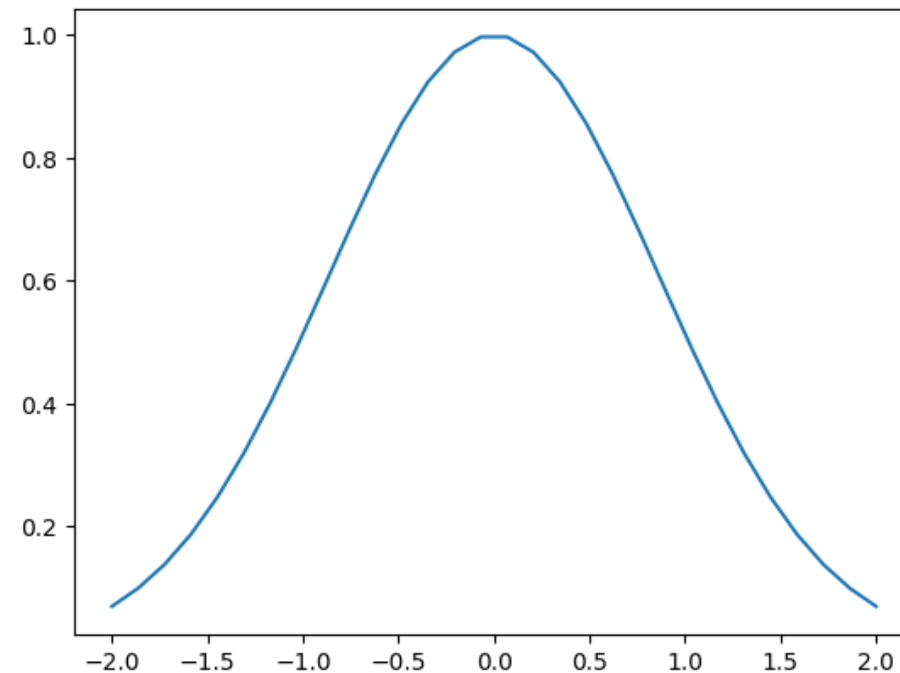
Prøv at printe x og y
og forklar hvad linspace og sin gør.



matplotlib

```
x = np.linspace(-2, 2, 30)  
y = np.exp(-x**2 / 1.5)  
plt.plot(x, y)  
plt.show()
```

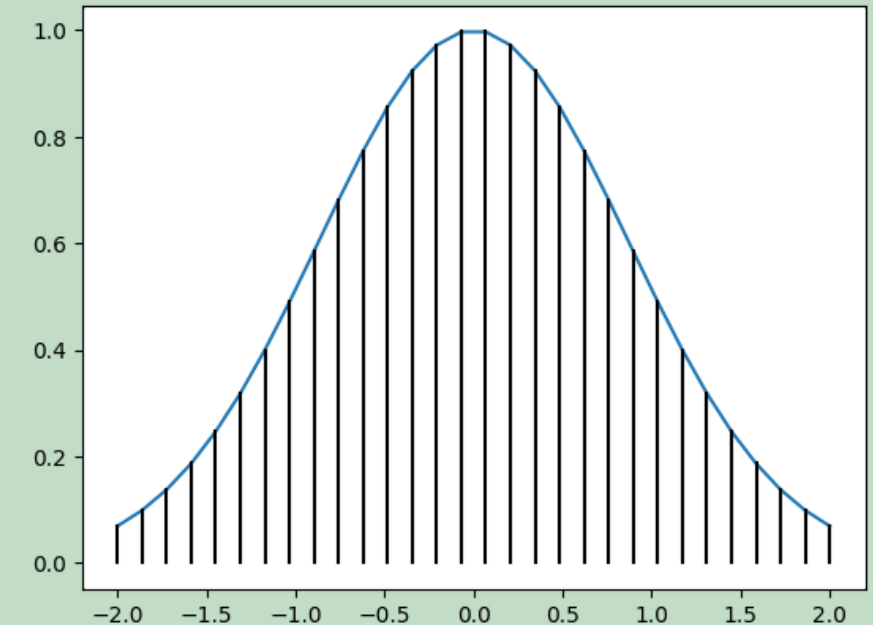
$$y = e^{-\frac{x^2}{1.5}}$$



np.trapz: Trapezoid-summing

- Beregn bestemt integrale ved at beregne arealet under kurven

```
x = np.linspace(-2, 2, 30)
y = np.exp(-x**2 / 1.5)
area = np.trapz(y, x)
print(area)      2.1248007109797644
```



scipy: Bestemt integration af funktion

```
x = np.linspace(-2, 2, 30)
y = np.exp(-x**2 / 1.5)
area = np.trapz(y, x)
print(area)      2.1248007109797644
```

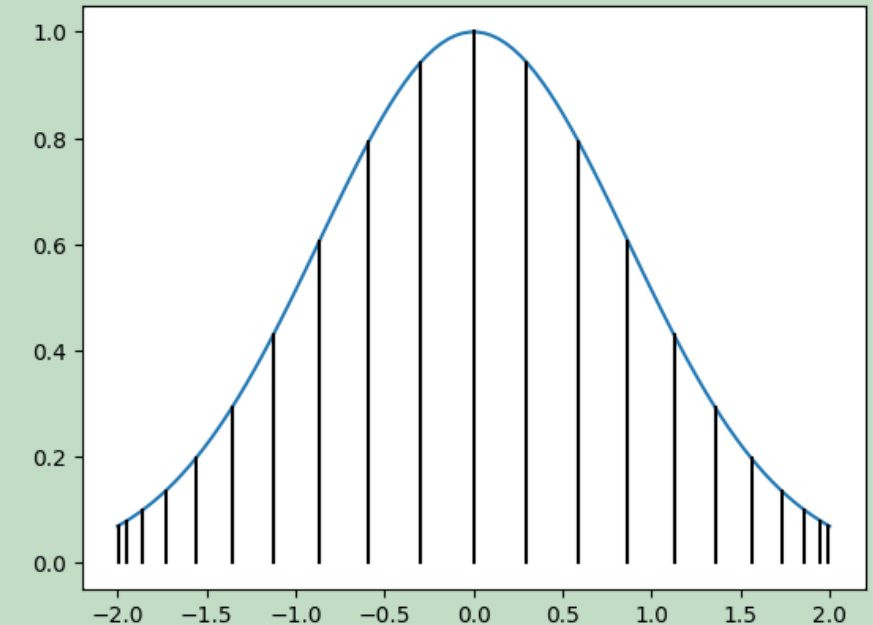
```
import scipy.integrate
```

```
def f(x):
    return np.exp(-x**2 / 1.5)
```

```
area, error = scipy.integrate.quad(f, -2, 2)
print(area, error)  2.1253876501824167 ± 2.09 × 10-9
```

- Totalt integral over hele $(-\infty, +\infty)$:

```
area, error = scipy.integrate.quad(f, -np.inf, np.inf)
print(area, error)  2.1708037636748040 ± 8.15 × 10-9
```



Så er det CSV-tid

- CSV: comma-separated values
- Simpelt tekstbaseret dataformat
- Eksempel: LoggerPro data for det skrå kast
- Download fra: m.strova.dk/python

”Plain text” i skraa.csv

```
x,y
0.0633,0.342
0.171,0.437
0.278,0.519
0.386,0.589
0.494,0.652
0.601,0.696
0.709,0.740
0.810,0.766
0.918,0.785
1.02,0.785
1.13,0.785
1.23,0.766
1.34,0.740
1.45,0.703
```

Materiale sammenskrevet af:

Brian M.V. Olesen

Haslev Gymnasium og HF
Juli 2009

05-07-2009 18:42

6.Gem filen

Du skal gemme LoggerPro filer for hver af dine analyser af dine kast. Filen gemmes på det relevante drev.

Husk at gemme filen med angivelse af dit navn og nummer på kastet, f.eks.: ”JENS PEDER # 1”

Dataoverførsel fra LoggerPro til TI-Nspire

Data fra LoggerPro filerne kopieres til hver sit TI-Nspire dokument. Markér og kopiere (t, x, y) data i LoggerPro filen ved at klikke og trække hen over cellerne. Indsæt de kopierede data i en **Lister og Regneark** applikation i TI-Nspire. Herefter navngives listerne henholdsvis *tdata*, *xdata* og *ydata*.

VideoAnalysis			
	Time (s)	X (m)	Y (m)
1	-3.33E-006	0.0633	0.342
2	0.0333	0.171	0.437
3	0.0667	0.278	0.519
4	0.100	0.386	0.589
5	0.133	0.494	0.652
6	0.167	0.601	0.696
7	0.199997	0.709	0.740
8	0.233	0.810	0.766
9	0.267	0.918	0.785
10	0.300	1.02	0.785
11	0.333	1.13	0.785
12	0.367	1.23	0.766
13	0.400	1.34	0.740
14	0.433	1.45	0.703

t	xdata	ydata
1	-3.33333	0.06329
2	0.033333	0.170883
3	0.066663	0.278475
4	0.099997	0.386068
5	0.13333	0.493661
6	0.166663	0.601254
7	0.199997	0.708847
8	0.23333	0.81644
9	0.266663	0.92403
10	0.299997	1.03162
11	0.33333	1.13921
12	0.366663	1.2468
13	0.399997	1.35439
14	0.43333	1.46198

t	xdata	ydata
1	-3.33333	0.06329
2	0.033333	0.170883
3	0.066663	0.278475
4	0.099997	0.386068
5	0.13333	0.493661
6	0.166663	0.601254
7	0.199997	0.708847
8	0.23333	0.81644
9	0.266663	0.92403
10	0.299997	1.03162
11	0.33333	1.13921
12	0.366663	1.2468
13	0.399997	1.35439
14	0.43333	1.46198

Bemærk at *tdata* er ”nulstillet”!

<https://www.uvmat.dk/mathit/skraa.pdf>

Tak til Brian M.V. Olesen, Haslev Gymnasium for data!

SCALGO
Let's create space for water

Så er det CSV-tid

Husk! Google Colab er ikke gratis! Brug rigtig Python til jeres studie!

Warning

Ensure that your files are saved elsewhere. This runtime's files will be deleted when this runtime is terminated. [More info](#)

OK

Download skraa.csv fra m.strova.dk/python



Untitled3.ipynb ☆

File Edit View Insert Runtime Tools Help All changes saved



Files



{x}



..



skraa.csv



+ Code + Text



```
import pandas as pd
```

```
df = pd.read_csv('skraa.csv')  
df.describe()
```



x

y



count	14.000000	14.000000
mean	0.757164	0.664643
std	0.444488	0.141647
min	0.063300	0.342000
25%	0.413000	0.604750
50%	0.759500	0.721500
75%	1.102500	0.766000
max	1.450000	0.785000

print(...) i notebooks

- Google Colab og Jupyter Notebook printer automatisk den sidste linje
- Det kan se forskelligt ud afhængig af om man skriver print(...)

```
import pandas as pd

df = pd.read_csv('skraa.csv')
df.describe()
```

	x	y
count	14.000000	14.000000
mean	0.757164	0.664643
std	0.444488	0.141647
min	0.063300	0.342000
25%	0.413000	0.604750
50%	0.759500	0.721500
75%	1.102500	0.766000
max	1.450000	0.785000

```
import pandas as pd

df = pd.read_csv('skraa.csv')
print(df.describe())
```

	x	y
count	14.000000	14.000000
mean	0.757164	0.664643
std	0.444488	0.141647
min	0.063300	0.342000
25%	0.413000	0.604750
50%	0.759500	0.721500
75%	1.102500	0.766000
max	1.450000	0.785000

- For at printe flere ting, brug print(...) eller brug "Split cell at cursor"

```
import matplotlib.pyplot as plt
```

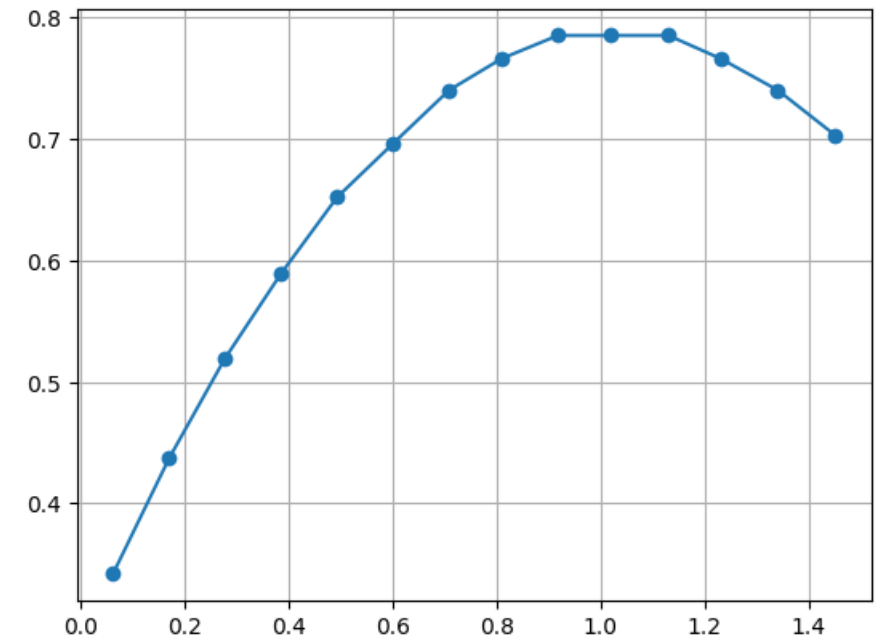
```
x = df['x']
```

```
y = df['y']
```

```
plt.plot(x, y, "o-")
```

```
plt.grid()
```

```
plt.show()
```



```
import matplotlib.pyplot as plt
```

```
x = df['x']
```

```
y = df['y']
```

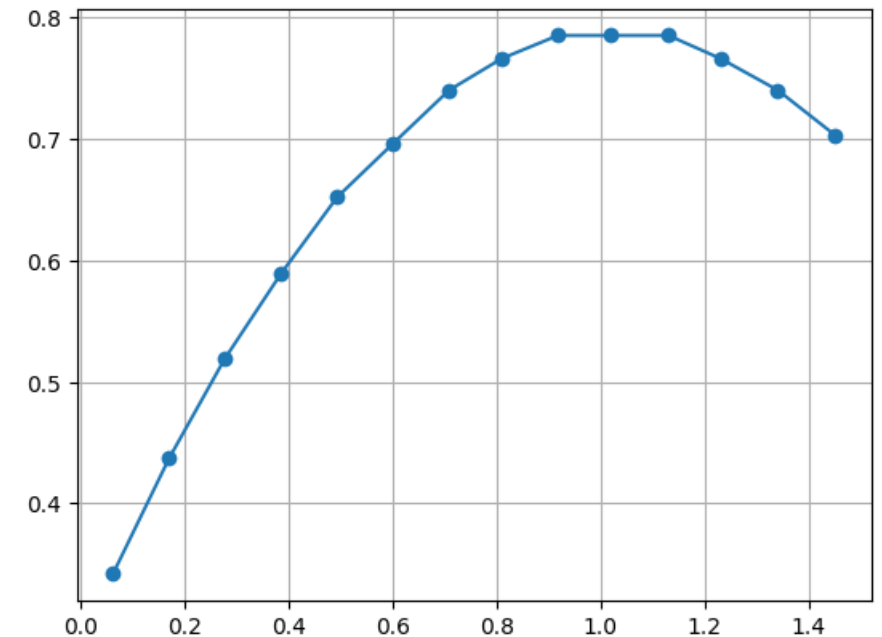
```
plt.plot(x, y, "o-")
```

```
plt.grid()
```

```
plt.show()
```

o: plot med store prikker på data

-: plot med en linje til at forbinde data



```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

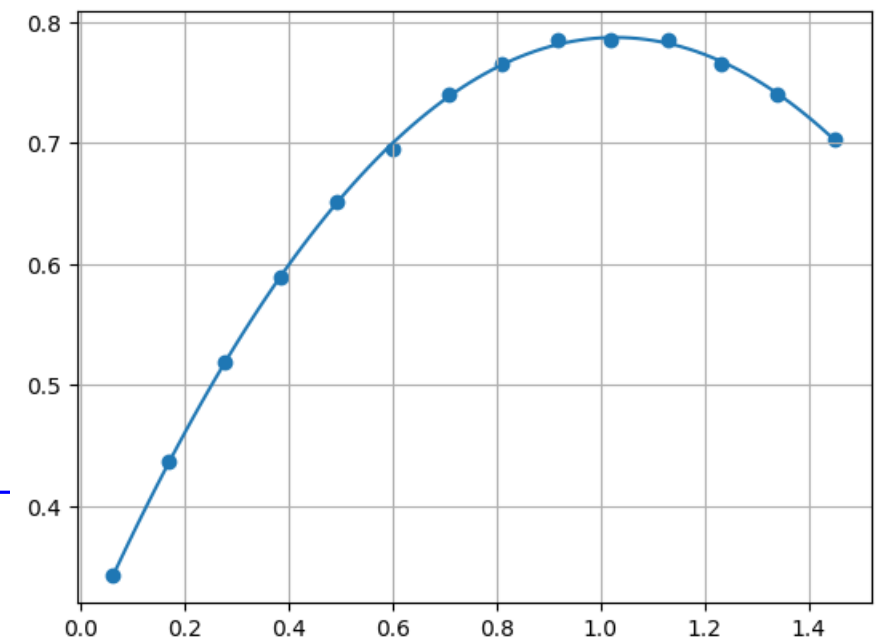
```
df = pd.read_csv('skraa.csv')
(a, b, c) = np.polyfit(df["x"], df["y"], 2)
print(f"Fit: y = {a:.4f} x^2 + {b:.4f} x + {c:.4f}")
```

1. Beregn fit

```
x_plot = np.linspace(df["x"].min(), df["x"].max(), 100)
y_plot = a * x_plot ** 2 + b * x_plot + c
```

2. Plot fit

```
plt.plot(x_plot, y_plot)
plt.scatter(df["x"], df["y"])
plt.grid()
plt.show()
```



```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

`{c:.4f}`: Skriv c med 4 cifre efter kommaet
`{c:.4}`: Skriv c med 4 betydende cifre

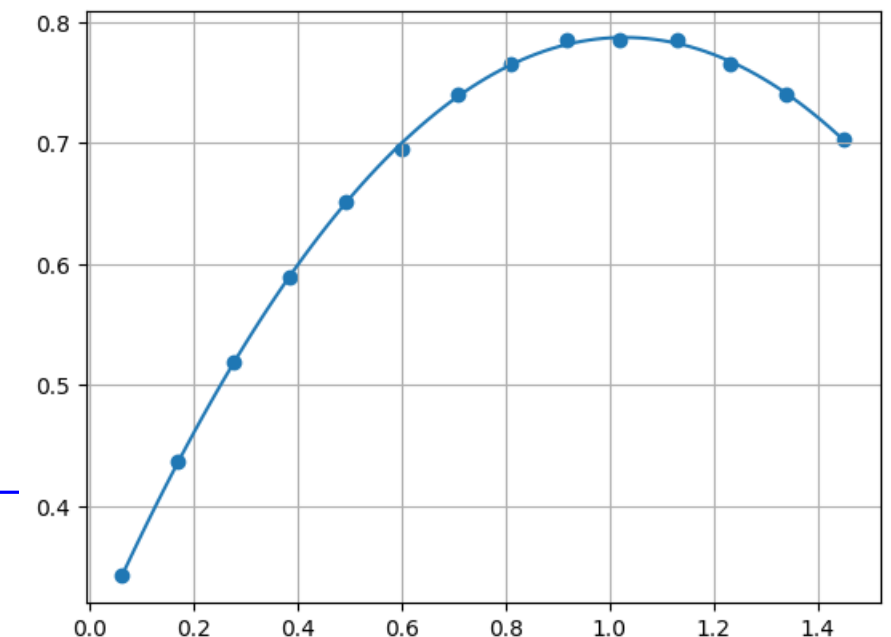
```
df = pd.read_csv('skraa.csv')
(a, b, c) = np.polyfit(df["x"], df["y"], 2)
print(f"Fit: y = {a:.4f} x^2 + {b:.4f} x + {c:.4f}")
```

1. Beregn fit

```
x_plot = np.linspace(df["x"].min(), df["x"].max(), 100)
y_plot = a * x_plot ** 2 + b * x_plot + c
```

2. Plot fit

```
plt.plot(x_plot, y_plot)
plt.scatter(df["x"], df["y"])
plt.grid()
plt.show()
```



Opgave

Brug $x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$ til at finde de 2 skæringer med x -aksen

- b^2 er `b ** 2`, og $\sqrt{\dots}$ er `np.sqrt( ... )`
- Hvordan håndteres $\pm$? Hvordan ville I gøre på en lommeregner?

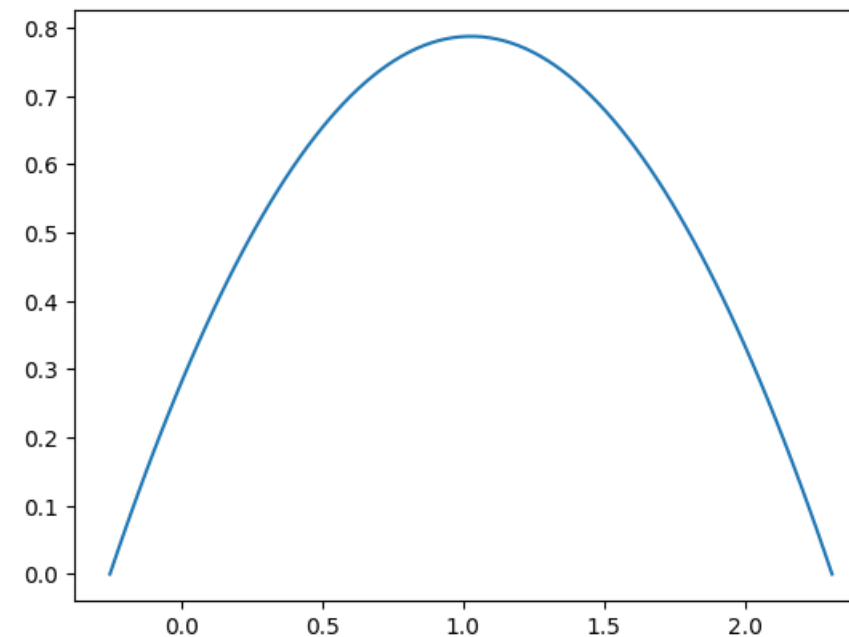
- Mulig løsning:

```
x1 = (-b + np.sqrt(b ** 2 - 4 * a * c)) / (2 * a)
x2 = (-b - np.sqrt(b ** 2 - 4 * a * c)) / (2 * a)
print(f"Crossings: x1 = {x1:.4f}, x2 = {x2:.4f}")
```

Opgave, del 2

- Gem de to skæringer som x1 og x2
 - Plot parablen mellem x1 og x2
- Mulig løsning:

```
x = np.linspace(x1, x2, 100)
y = a * x ** 2 + b * x + c
plt.plot(x, y)
plt.show()
```



Opgave, del 3

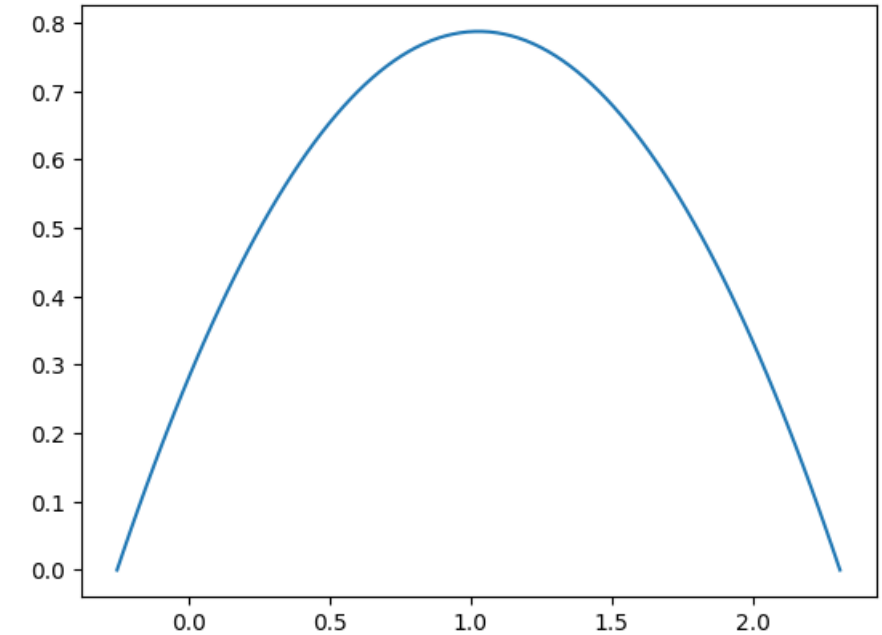
- Beregn arealet mellem x -aksen og kurven

- Mulig løsning: Brug `scipy.integrate.quad`

```
import scipy.integrate
```

```
def f(x):  
    return a * (x**2) + b * x + c
```

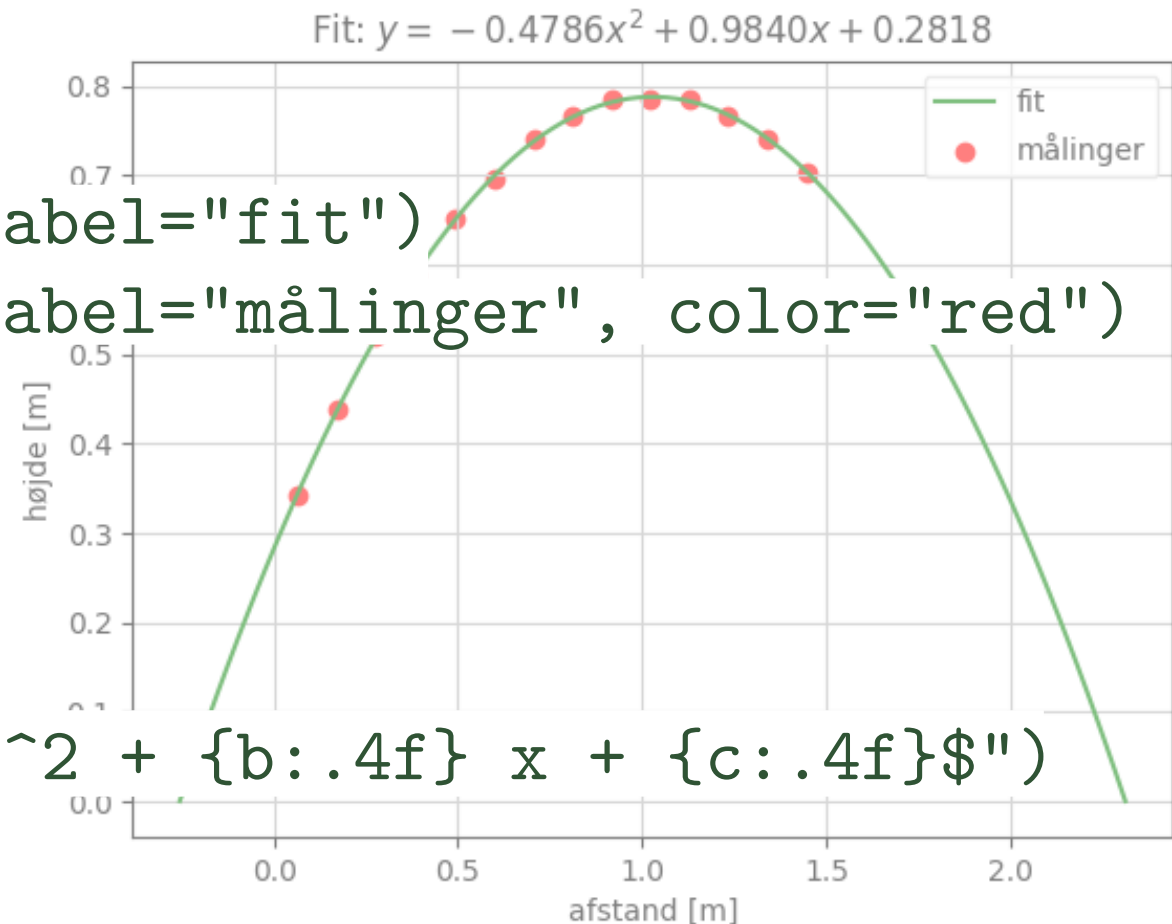
```
integral, error = scipy.integrate.quad(f, x1, x2)  
print(f"Integral: {integral:.4f} +/- {error:.4f}")
```



Et plot med det hele, tak

- Til rapporter skal I altid huske signaturforklaring og akse-labels

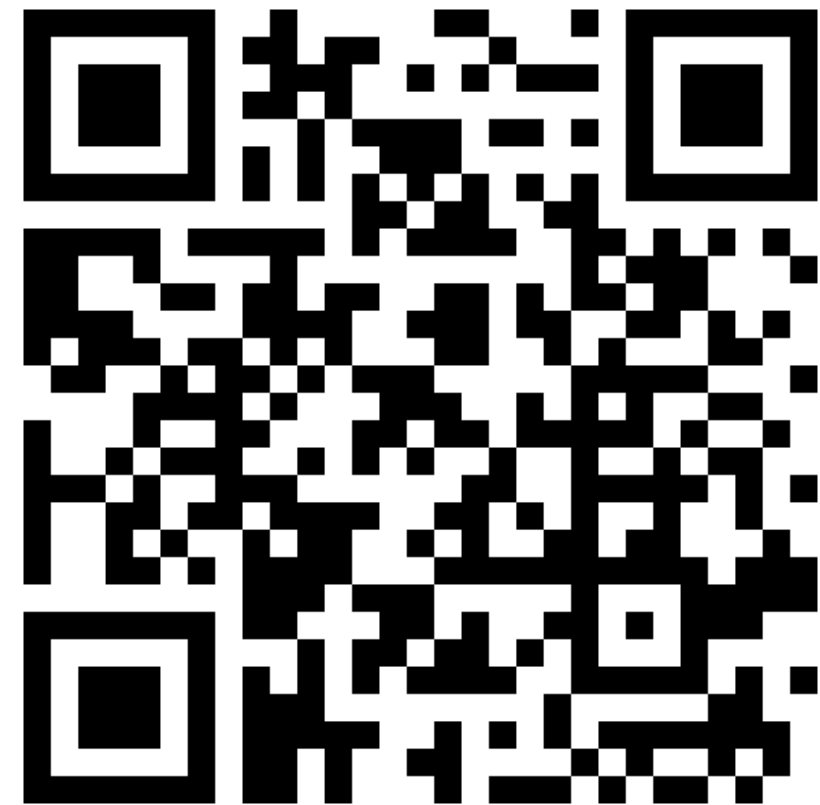
```
x = np.linspace(x1, x2, 100)
y = a * (x**2) + b * x + c
plt.plot(x, y, color="green", label="fit")
plt.scatter(df["x"], df["y"], label="målinger", color="red")
plt.grid()
plt.legend()
plt.xlabel("afstand [m]")
plt.ylabel("højde [m]")
plt.title(f"Fit: $y = {a:.4f} x^2 + {b:.4f} x + {c:.4f}$")
plt.show()
```



Python for begyndere
`m.strova.dk/python`

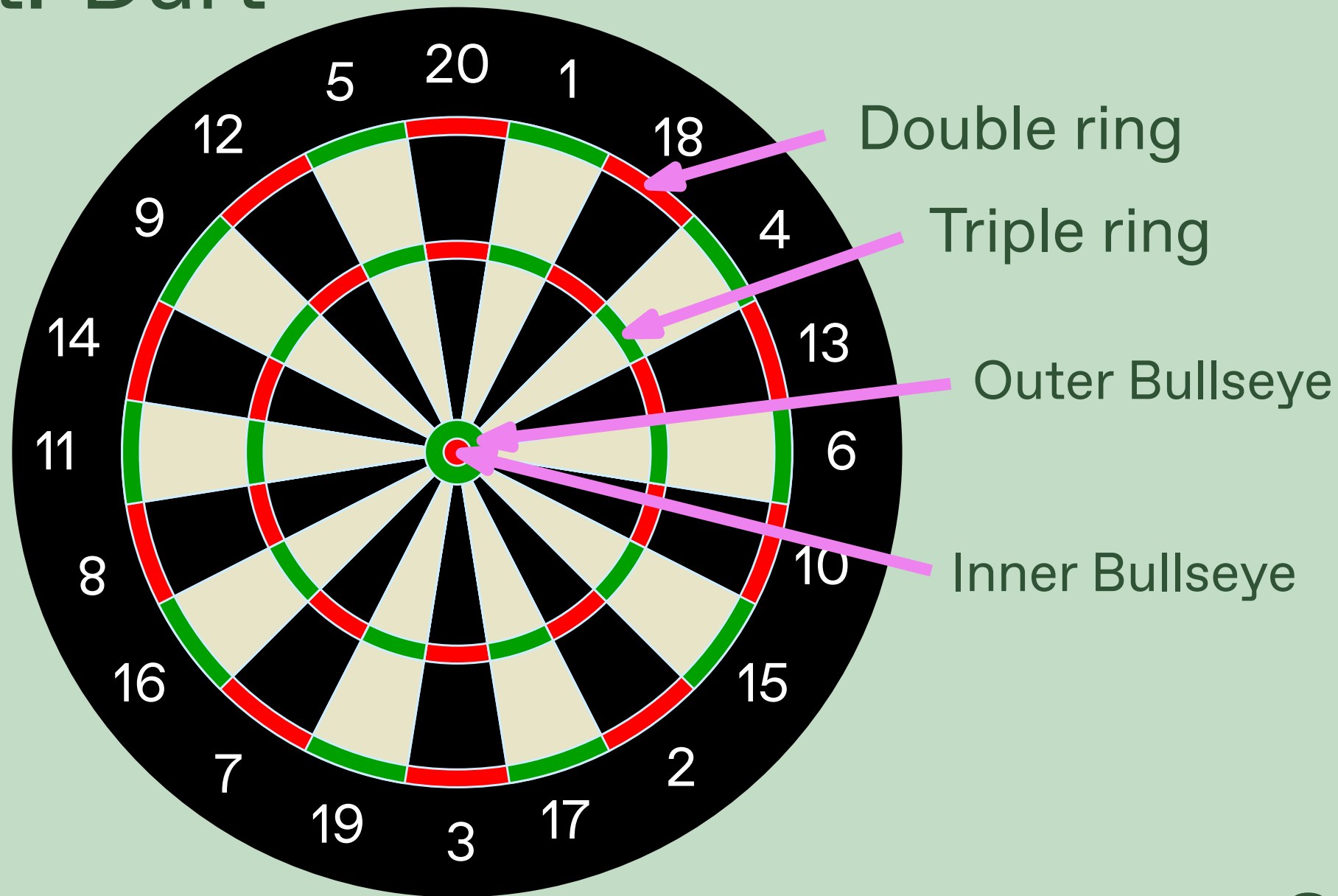
Mathias Rav
`mathias@scalgo.com`

SCALGO



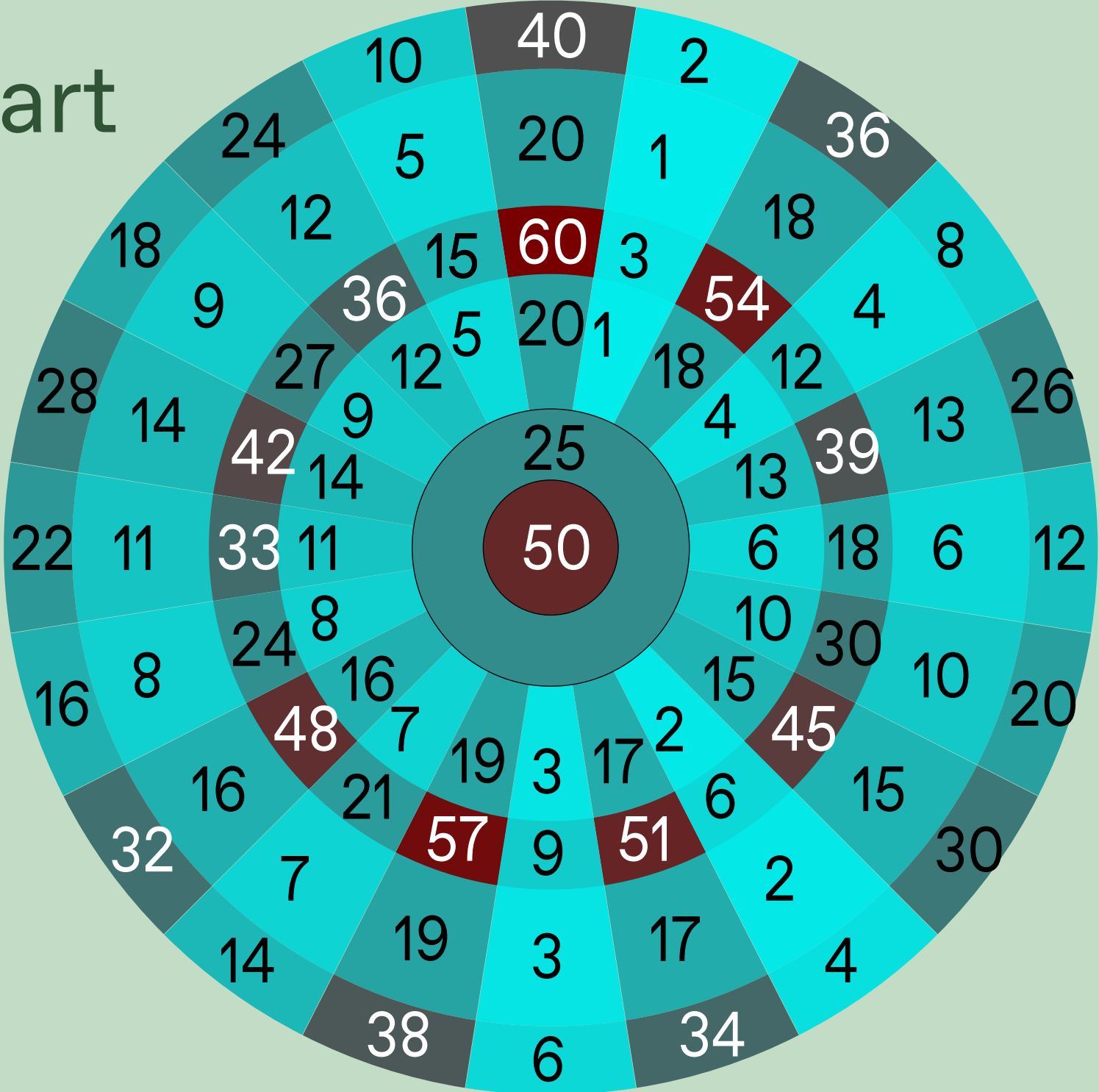
evaluering: <https://forms.gle/WKVFVMpPy4wWNJAU7>

Projekt: Dart



Dartboard_diagram.svg CC-BY-SA Tijmen Stam, from Wikimedia Commons

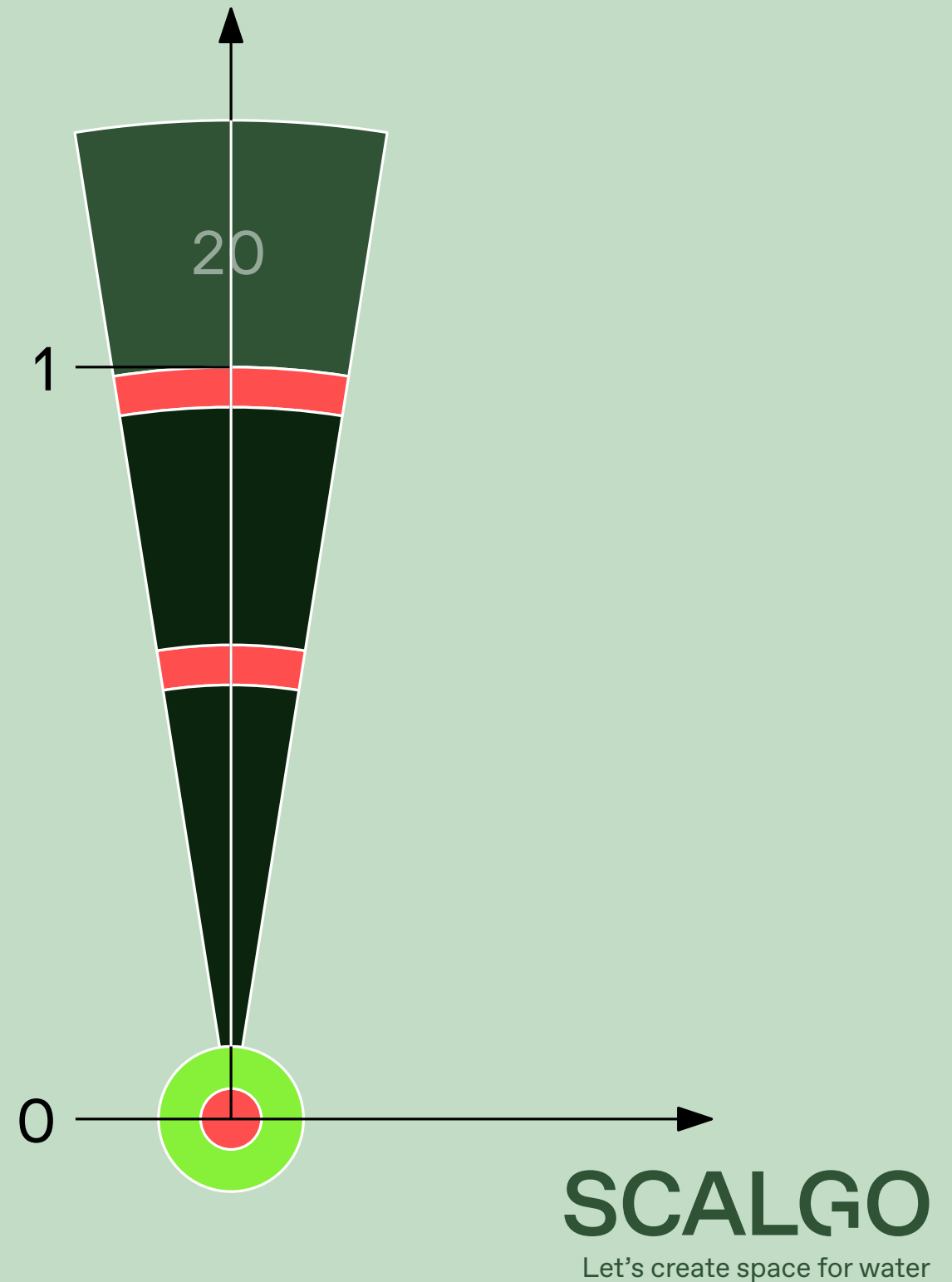
Projekt: Dart



Dartboard_heatmap.svg CC-BY-SA Cmglee, from Wikimedia Commons

Projeckt: Dart

Model: Dartskiven er en cirkelskive med radius 1 og bullseye i $(0, 0)$



Et enkelt skud: Inden for skiven?

```
import numpy as np

x = np.random.normal(0.0, 1.0)
y = np.random.normal(0.0, 1.0)

r = np.sqrt((x ** 2) + (y ** 2))
phi = np.arctan2(y, x)

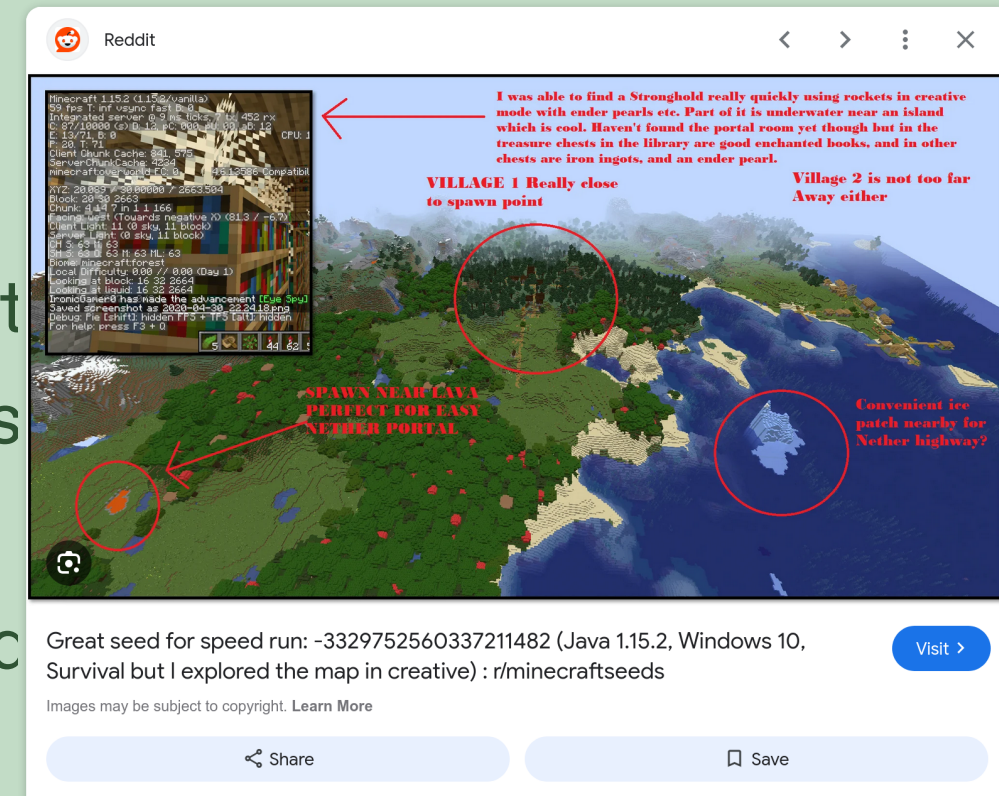
r < 1
```

Tilfældige tal

- Der er meget lidt i universet der er ægte tilfældigt
- En computer er som udgangspunkt deterministisk
- Eksterne input kan være svære at forudsige
 - Timing på harddisk, museklik, netværksforbindelser
- Computeren bruger eksterne inputs som "seed"

Tilfældige tal

- Der er meget lidt i universet der er ægte tilfældigt
- En computer er som udgangspunkt deterministisk
- Eksterne input kan være svære at forudsige
 - Timing på harddisk, museklik, netværksforbindelse
- Computeren bruger eksterne inputs som ”seed”
- Givet et seed, kan man lave en uendelig følge af pseudo-tilfældige tal
- $\text{Seed} \approx \text{sidetal}$ i Schaums tilfældige taltabel



Tilfældige tal, eller bare tal fra Schaums?

```
import numpy as np
```

```
np.random.seed(71120959)
```

```
x = np.random.normal(0.0, 1.0)
```

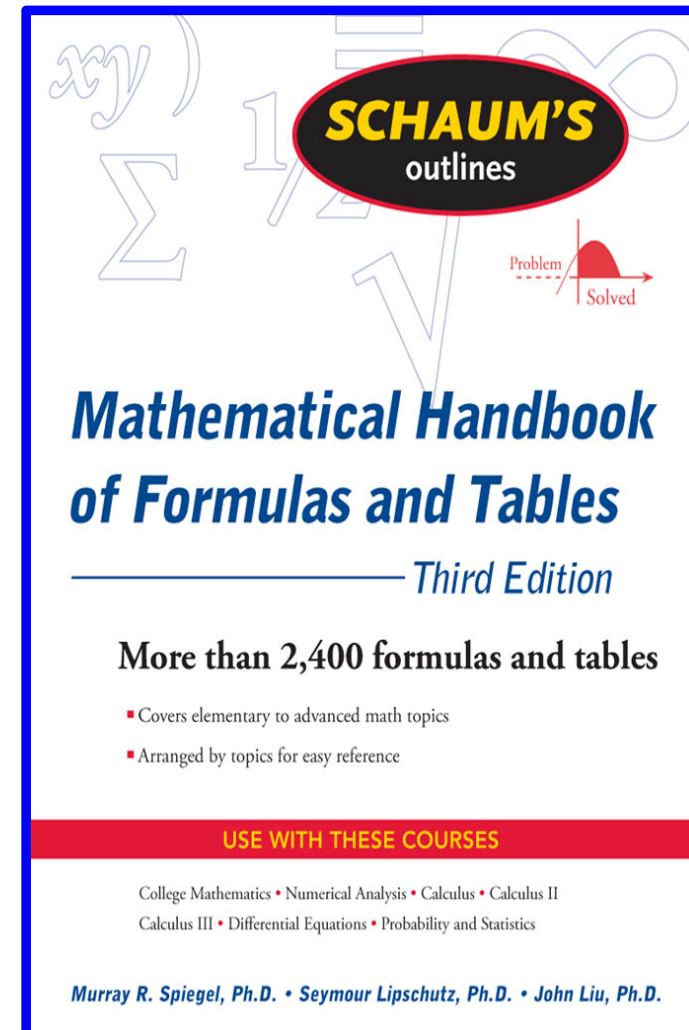
```
y = np.random.normal(0.0, 1.0)
```

```
print(x, y)
```

```
r = np.sqrt((x ** 2) + (y ** 2))
```

```
phi = np.arctan2(y, x)
```

```
r < 1
```



42 RANDOM NUMBERS															
51772	74640	42331	29044	46621	62898	93582	04186	19640	87056						
24033	23491	83587	06568	21960	21387	76105	10863	97453	90581						
46989	60173	52978	25424	11645	55870	56974	37428	93507	94271						
30586	02132	75797	45406	31041	86707	12973	17169	88116	42187						
03685	79353	81938	82322	96799	85659	36081	50884	14070	74950						
64987	03355	95863	20790	65304	55189	00745	65253	11822	15804						
16630	64759	51135	98527	62586	41889	25439	88036	24034	67283						
09448	56301	57683	30277	94623	85418	68829	06652	41982	49159						
21631	91157	77331	80710	52230	16835	48653	71590	16159	14676						
91097	17480	29414	06829	87843	28195	27279	47152	35683	47280						
50532	25496	95652	42457	78547	76552	50020	24819	52984	76168						
07136	40876	79971	54195	25708	51817	36732	72484	94923	75936						
27989	64728	10744	08396	56242	90985	28868	99431	50995	20507						
86184	73949	36601	46253	00477	25234	09908	36574	72139	70185						
54398	21154	97810	36764	32869	11785	55261	59009	38714	38723						
65544	34371	09591	07839	58892	92843	72828	91341	84821	63886						
08263	65952	85762	64236	29238	18776	84303	99247	46149	03259						
39817	67906	48236	16057	81812	15815	63700	85915	19219	45943						
62257	04077	79443	95203	02479	30763	92486	54083	23631	06825						
53298	90276	62545	21944	16530	03878	07516	95715	02526	33537						

Tæl skud inden for skiven

```
np.random.seed(202402535)

skud = np.random.normal(0.0, 1.0, size=(1000, 2))

x = skud[:, 0]

y = skud[:, 1]

r = np.sqrt((x ** 2) + (y ** 2))
phi = np.arctan2(y, x)

np.sum(r < 1)
```

Tæl skud inden for skiven

```
np.random.seed(202402535)
```

Vi laver en 1000×2 matrix

```
skud = np.random.normal(0.0, 1.0, size=(1000, 2))
```

```
x = skud[:, 0]
```

Første søjle

```
y = skud[:, 1]
```

Anden søjle

```
r = np.sqrt((x ** 2) + (y ** 2))
```

```
phi = np.arctan2(y, x)
```

```
np.sum(r < 1)
```

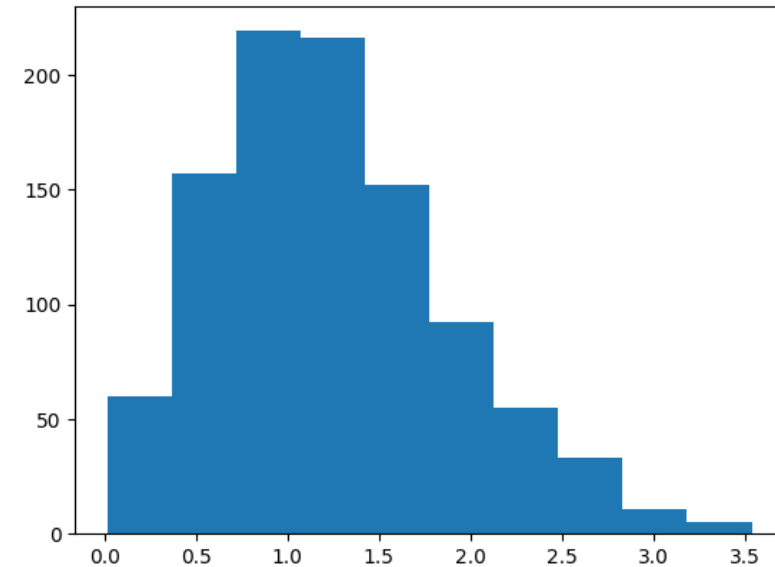
`np.sum` fortæller hvor mange der er True

Prøv at forklare output når man bare skriver: `r < 1`

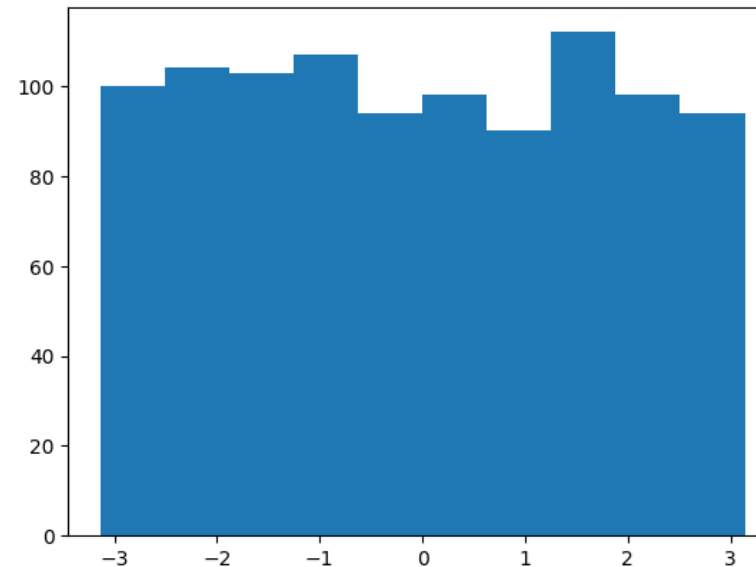
Histogram-plot af r og ϕ

```
import matplotlib.pyplot as plt
```

```
plt.hist(r)  
plt.show()
```

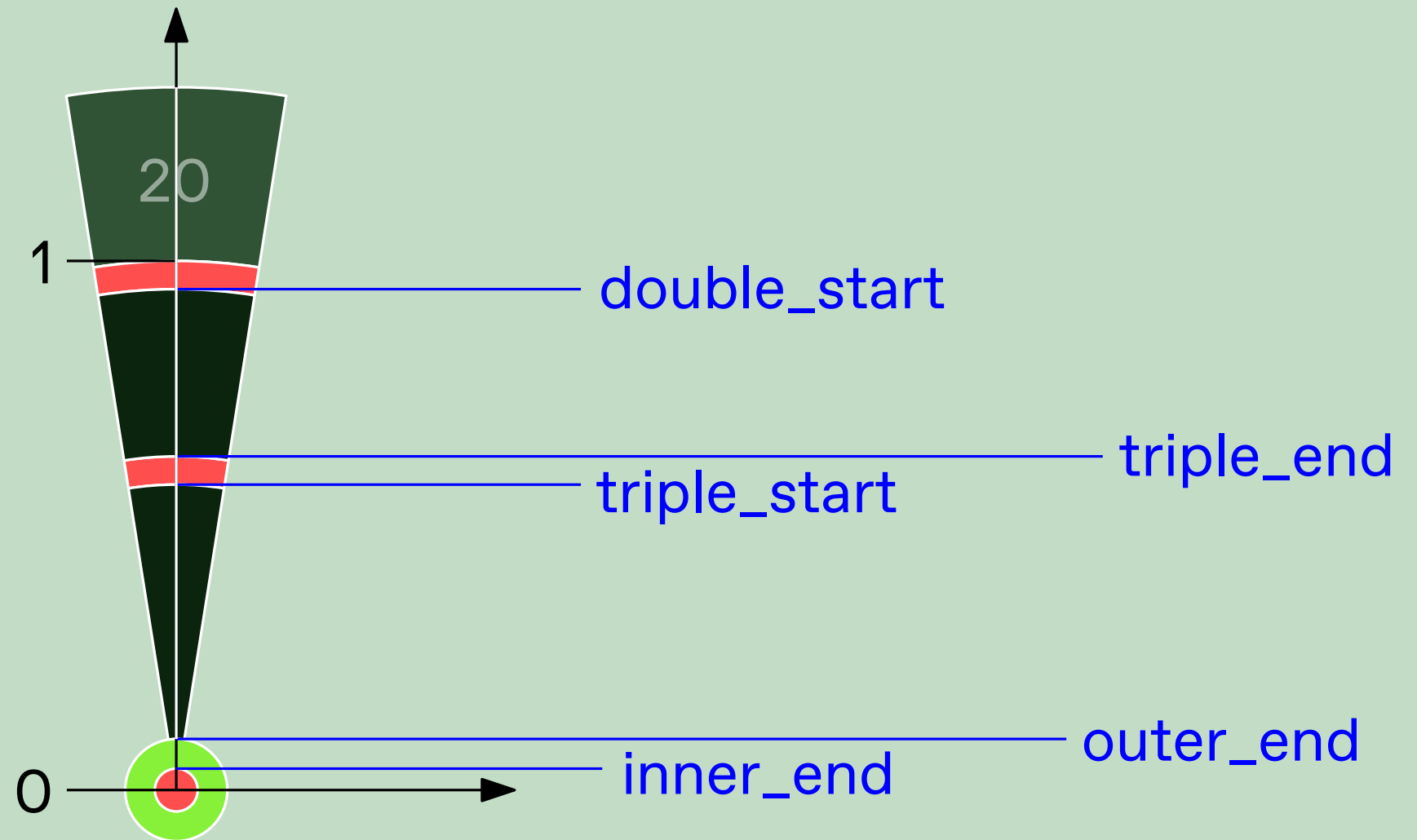


```
plt.hist(phi)  
plt.show()
```



Projeckt: Dart

```
double_start = 0.95  
triple_end = 0.63  
triple_start = 0.58  
outer_end = 0.10  
inner_end = 0.05
```



Husk: Dartskiven har radius 1
og bullseye er i (0, 0)

Bullseye

```
:
```

```
double_start = 0.95
```

```
triple_end = 0.63
```

```
triple_start = 0.58
```

```
outer_end = 0.10
```

```
inner_end = 0.05
```

```
inner_bullseye = r < inner_end
```

```
np.sum(inner_bullseye)
```

Hvor ofte rammer man bullseye?

- Matthew Edgar, professionel dartspiller:
64 ud af 300 pile



youtu.be/EmByS3gZnvM

```
np.random.seed(202402535)
stddev = 0.1
skud = np.random.normal(0.0, stddev, size=(300, 2))
x = skud[:, 0]
y = skud[:, 1]
r = np.sqrt((x ** 2) + (y ** 2))
np.sum(r < inner_end)
```

Hvor ofte rammer man bullseye?

- Matthew Edgar, professionel darts spiller:
64 ud af 300 pile



youtu.be/EmByS3gZnvM

```
np.random.seed(202402535)
```

```
stddev = 0.1
```

Spredning på normalfordeling

```
skud = np.random.normal(0.0, stddev, size=(300, 2))
```

300 skud

```
x = skud[:, 0]
```

```
y = skud[:, 1]
```

```
r = np.sqrt((x ** 2) + (y ** 2))
```

```
np.sum(r < inner_end)
```

Afstandsmål

```
double_start = 0.95  
triple_end = 0.63  
triple_start = 0.58  
outer_end = 0.10  
inner_end = 0.05
```

```
inner_bullseye = r < inner_end  
outer_bullseye = (inner_end < r) & (r < outer_end)  
triple = (triple_start < r) & (r < triple_end)  
double = (double_start < r) & (r < 1.0)
```

```
print(np.mean(inner_bullseye), np.mean(outer_bullseye),  
      np.mean(triple), np.mean(double))
```

Afstandsmål

```
double_start = 0.95  
triple_end = 0.63  
triple_start = 0.58  
outer_end = 0.10  
inner_end = 0.05
```

np.mean af True/False-array:
Andel af array som er True



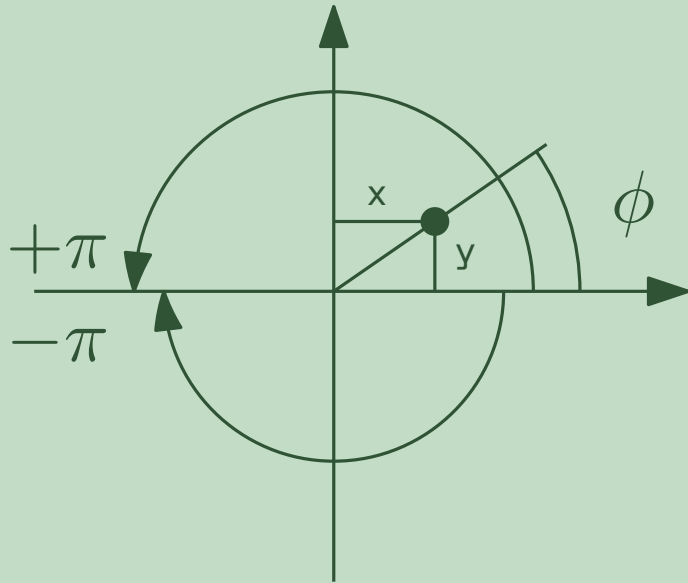
&-tegn bruges med numpy-arrays



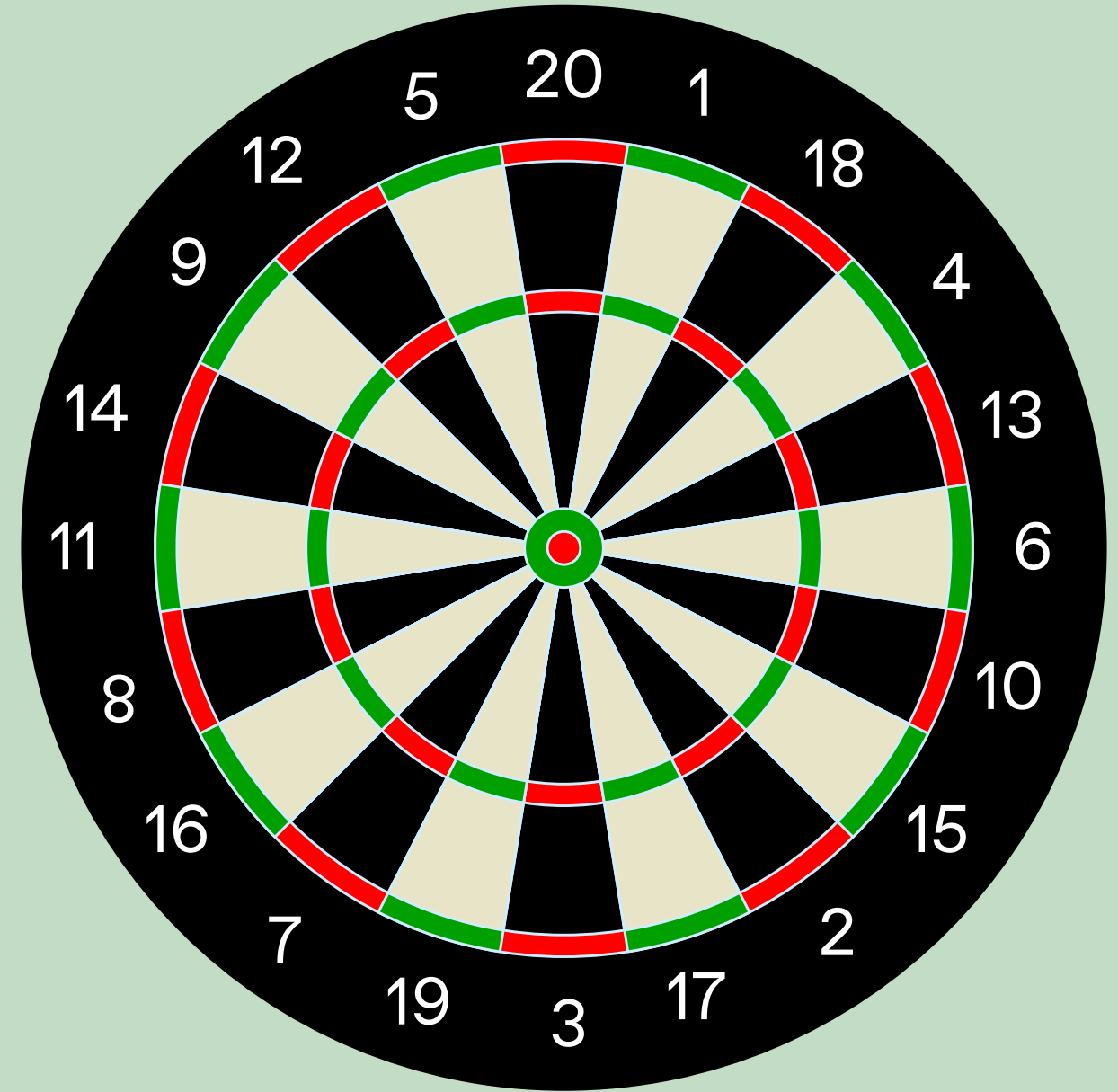
```
inner_bullseye = r < inner_end  
outer_bullseye = (inner_end < r) & (r < outer_end)  
triple = (triple_start < r) & (r < triple_end)  
double = (double_start < r) & (r < 1.0)  
  
print(np.mean(inner_bullseye), np.mean(outer_bullseye),  
      np.mean(triple), np.mean(double))
```

Point

```
phi = np.arctan2(y, x)
```



Hvordan finder man hvilken af de 20 pointskiver man har ramt?



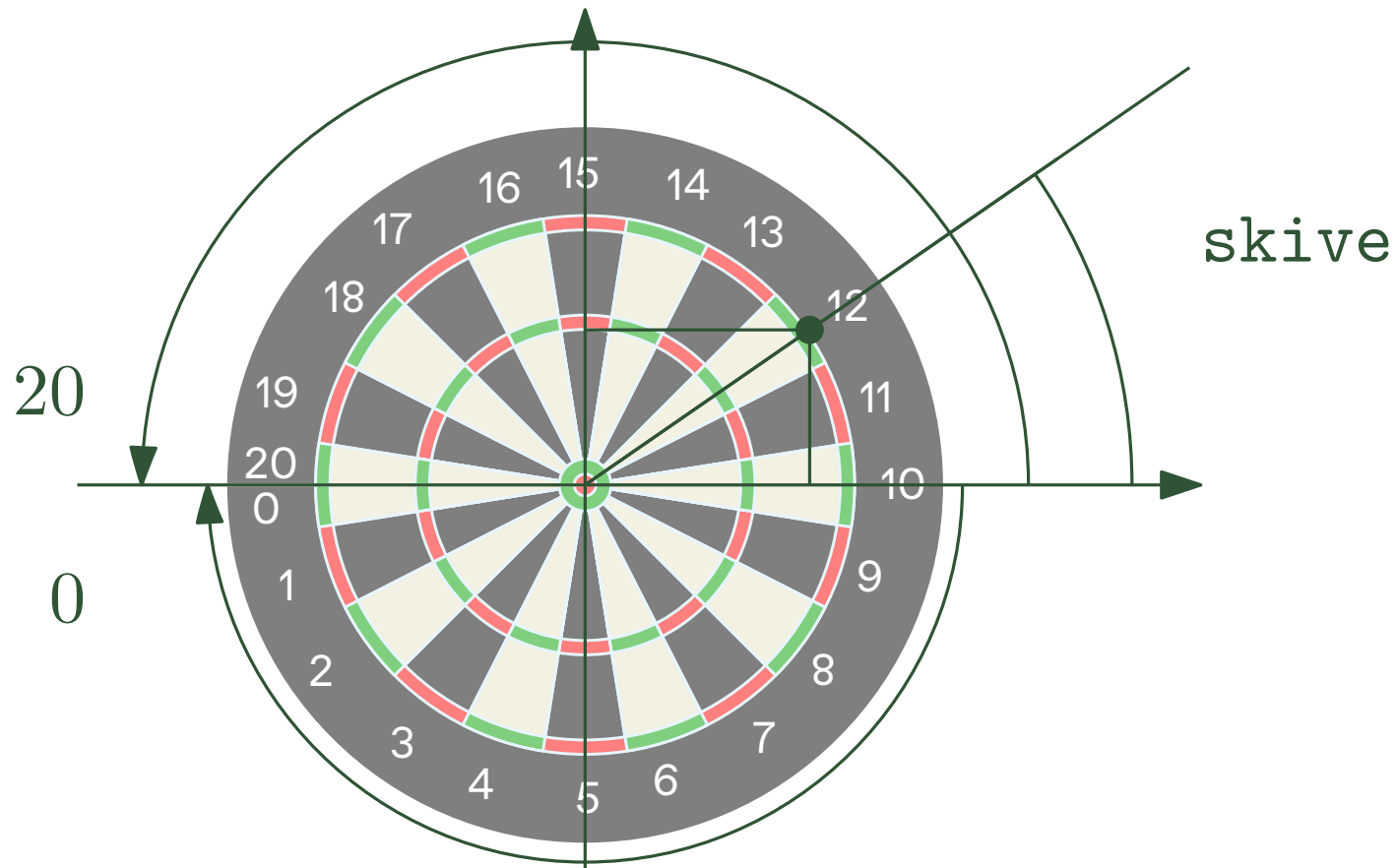
Dartboard_diagram.svg CC-BY-SA Tijmen Stam, from Wikimedia Commons

Point

```
skive = np.round((phi / np.pi) * 10) + 10
```

```
skive[skive == 20] = 0
```

Find alle steder hvor skive er 20,
og sæt dem til 0 i stedet



Dartboard_diagram.svg CC-BY-SA Tijmen Stam, from Wikimedia Commons

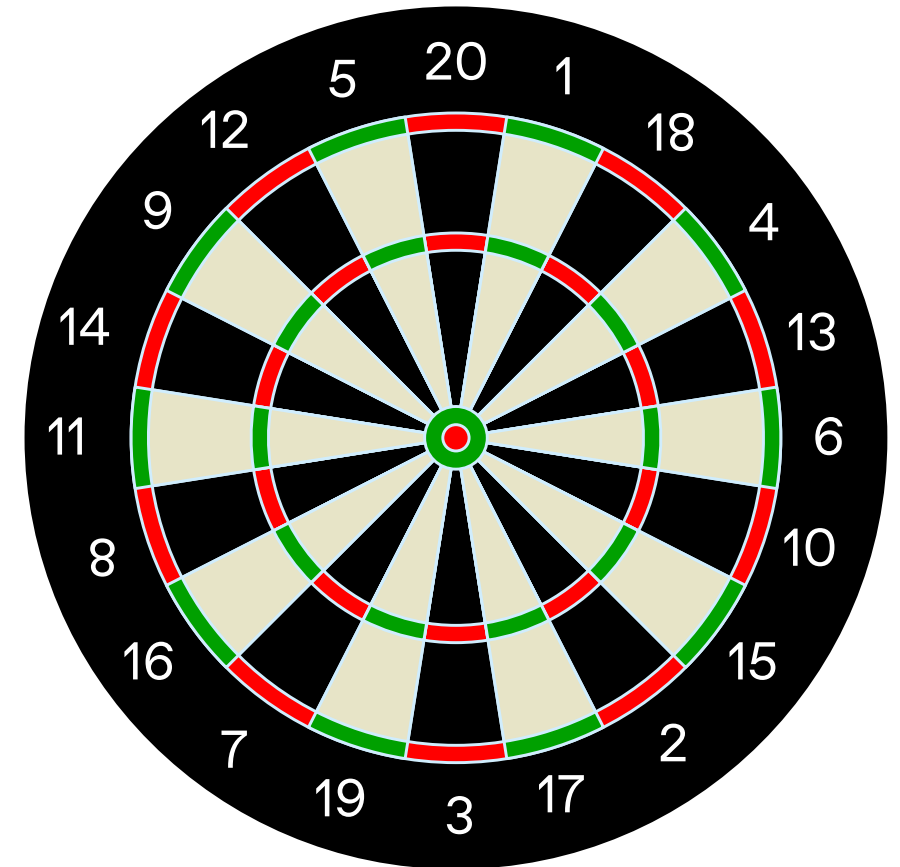
Point

```
skive = np.round((phi / np.pi) * 10) + 10
```

```
skive[skive == 20] = 0
```

```
skivepoint = np.array([  
    11, 8, 16, 7, 19, 3, 17, 2, 15, 10,  
    6, 13, 4, 18, 1, 20, 5, 12, 9, 14  
])
```

```
point = skivepoint[skive]
```



Dartboard_diagram.svg CC-BY-SA Tijmen Stam, from Wikimedia Commons

Point

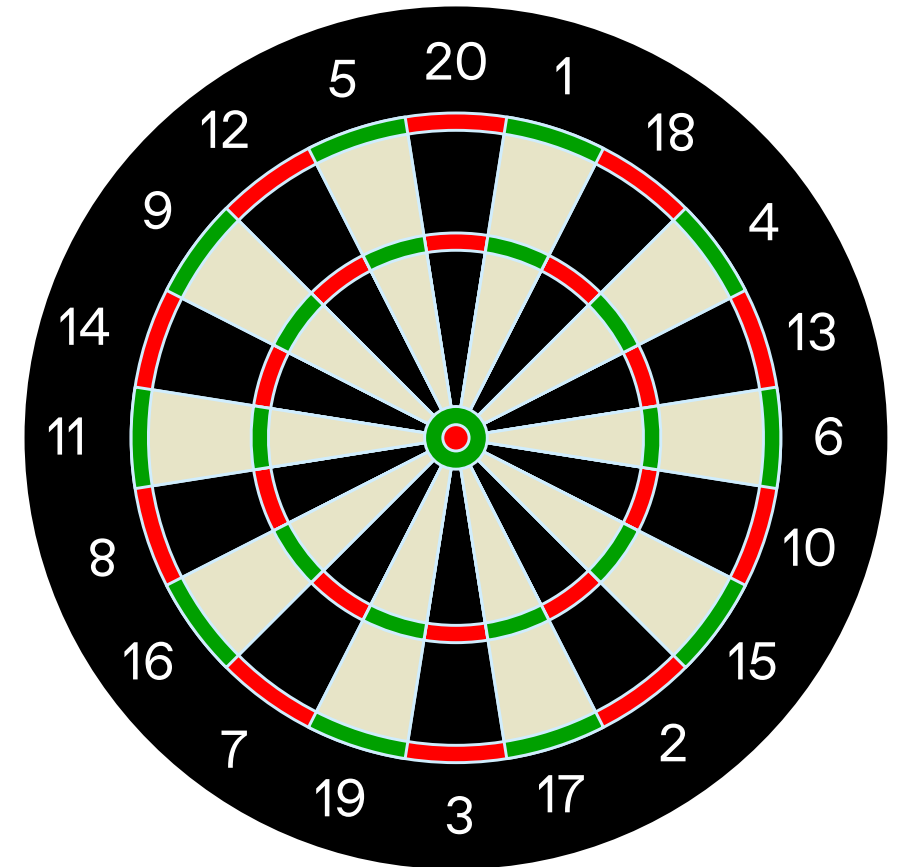
```
skive = np.round((phi / np.pi) * 10) + 10
```

```
skive[skive == 20] = 0
```

```
skivepoint = np.array([  
    011, 18, 216, 37, 419, 53, 617, 72, 815, 910,  
    106, 1113, 124, 1318, 141, 1520, 165, 1712, 189, 1914  
])
```

```
point = skivepoint[skive]
```

tal i skive er indekser i skivepoint



Dartboard_diagram.svg CC-BY-SA Tijmen Stam, from Wikimedia Commons

arh, hvad er der nu galt

0s

```
skive = np.round((phi / np.pi) * 10) + 10
skive[skive == 20] = 0
skivepoint = np.array([
    11, 8, 16, 7, 19, 3, 17, 2, 15, 10,
    6, 13, 4, 18, 1, 20, 5, 12, 9, 14
])
point = skivepoint[skive]
```



```
-----
IndexError                                Traceback (most recent call last)
<ipython-input-26-b9d661e7b805> in <cell line: 7>()
      5     6, 13, 4, 18, 1, 20, 5, 12, 9, 14
      6 ])
----> 7 point = skivepoint[skive]

IndexError: arrays used as indices must be of integer (or boolean) type
```

- "must be of integer type"... `print(skive.dtype)` → `float64`

.astype(int) to the rescue

```
skive = (np.round((phi / np.pi) * 10) + 10).astype(int)
skive[skive == 20] = 0
skivepoint = np.array([
    11, 8, 16, 7, 19, 3, 17, 2, 15, 10,
    6, 13, 4, 18, 1, 20, 5, 12, 9, 14
])
point = skivepoint[skive]
```

Nu mangler vi bare bulls, doubles, triples

Bulls, doubles, triples

```
point[r > 1] = 0
```

```
point[double] *= 2
```

```
point[triple] *= 3
```

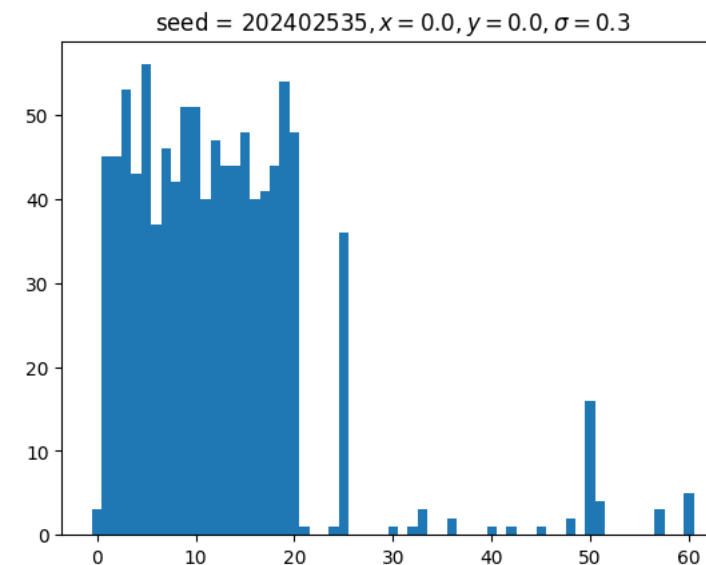
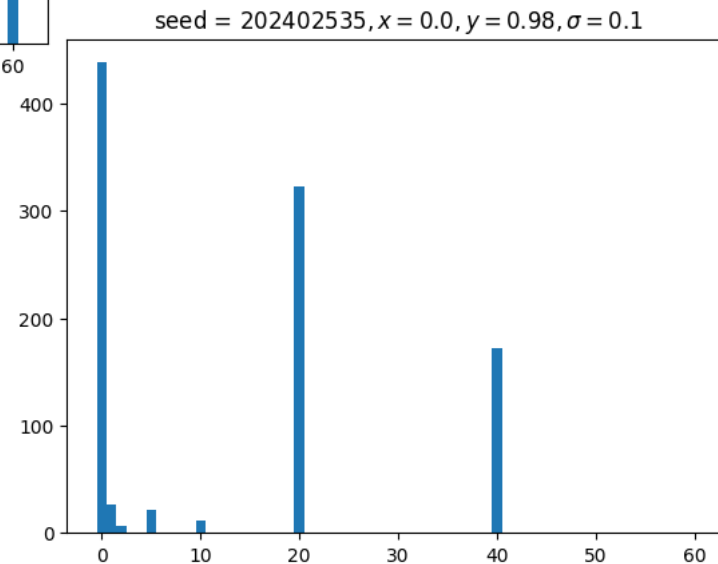
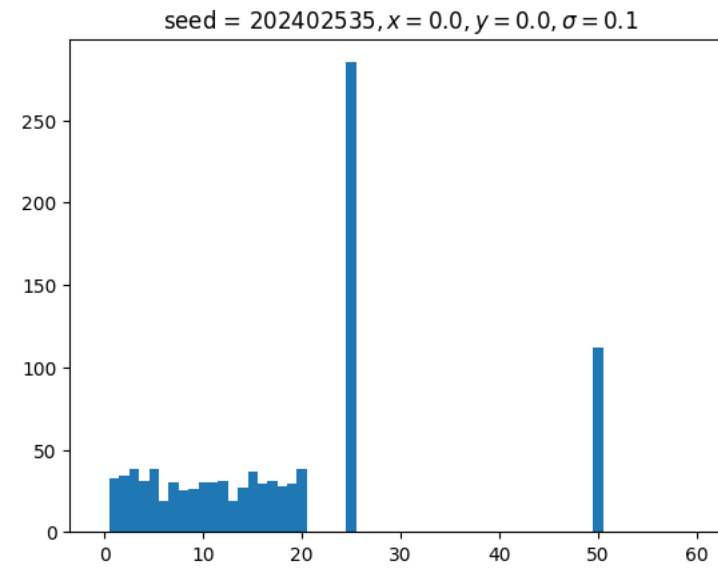
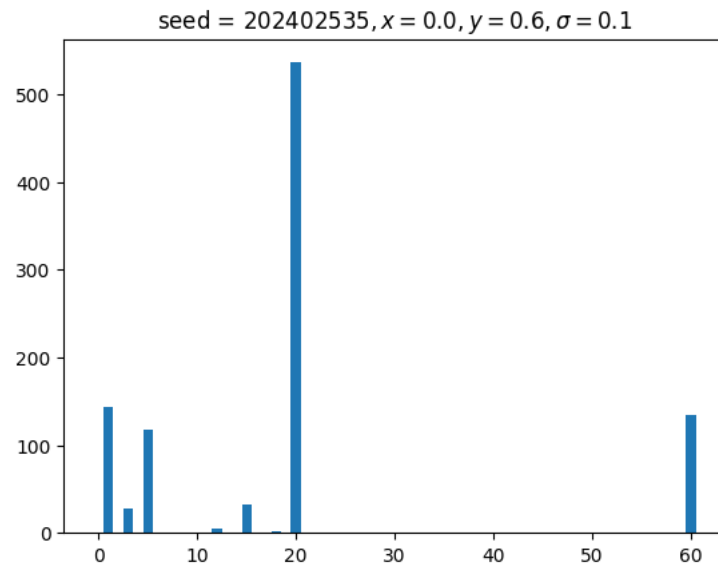
```
point[outer_bullseye] = 25
```

```
point[inner_bullseye] = 50
```

```
print(np.mean(point))
```

Histogram-plot

```
plt.title(f"seed = ${seed}, x = {target_x}, y = {target_y}, \sigma = {stddev}$")
plt.hist(point, bins=np.arange(0, 62) - 0.5)
plt.show()
```



Gem som csv

```
import pandas as pd
```

```
df = pd.DataFrame({  
    'x': x,  
    'y': y,  
    'r': r,  
    'phi': phi,  
    'point': point  
})
```

```
df.to_csv('dart_data.csv', index=False)
```

```

stddev = 0.1
target_x = 0.0
target_y = 0.6

seed = 202402535
np.random.seed(seed)
skud = np.random.normal(0.0, stddev, size=(1000, 2))
x = skud[:, 0] + target_x
y = skud[:, 1] + target_y
r = np.sqrt((x ** 2) + (y ** 2))
phi = np.arctan2(y, x)

inner_bullseye = r < inner_end
outer_bullseye = (inner_end < r) & (r < outer_end)
triple = (triple_start < r) & (r < triple_end)
double = (double_start < r) & (r < 1.0)

skive = (np.round((phi / np.pi) * 10) + 10).astype(int)
skive[skive == 20] = 0
skivepoint = np.array([
    11, 8, 16, 7, 19, 3, 17, 2, 15, 10,
    6, 13, 4, 18, 1, 20, 5, 12, 9, 14
])
point = skivepoint[skive]

point[r > 1] = 0
point[double] *= 2
point[triple] *= 3
point[outer_bullseye] = 25
point[inner_bullseye] = 50
print(np.mean(point))
plt.hist(point, bins=np.arange(0, 62) - 0.5)
plt.title(f"seed = ${seed}, x = {target_x}, y = {target_y}, \sigma = {stddev}$")
plt.show()

df = pd.DataFrame({'x': x, 'y': y, 'r': r, 'phi': phi, 'point': point})
df.to_csv('dart_data.csv', index=False)

```

Appendix: More Python syntax

```
a = []  
for x in [1, 2, 3]:  
    a.append(x * 2)  
  
b = [x * 2 for x in [1, 2, 3]]
```

Appendix: More Python syntax

```
any([False, True, False])  
any([False, False, False])  
any([])
```

```
all([False, False, True])  
all([True, True, True])  
all([])
```

Appendix: More Python syntax

```
a = [7, 42, 13]
any([x == 42 for x in a])
all([x == 42 for x in a])
a == 42
```

```
b = np.array([7, 42, 13])
b == 42
np.any(b == 42)
np.all(b == 42)
```

Appendix: More Python syntax

```
for i, x in enumerate(a):  
    print(i, x)
```

```
for x, y in zip(a, b):  
    print(x, y)
```

```
for i, (x, y) in enumerate(zip(a, b)):  
    print(i, x, y)
```

Appendix: Beware of floats

```
0.1 + 0.2 + 0.3
```

```
(0.1 + 0.2) + 0.3
```

```
0.1 + (0.2 + 0.3)
```

```
x = 0.1 + 0.2
```

```
y = 0.3
```

```
x == y
```

```
print(f'{x:.30f}') # 30 decimals
```

```
print(f'{y:.30f}') # 30 decimals
```

Appendix: Strings

```
print("abc")  
print('de\'f')  
print("'ghi'")  
print("'jk\nl'\")  
print("mn\  
o")  
print("p\\q\\tr")  
print("\N{SNOWMAN}")
```

```
print('\let\epsilon\varepsilon') # \v = vertical tab  
print('\\let\\epsilon\\varepsilon') # many backslashes  
print(r'\let\epsilon\varepsilon') # more readable
```

Appendix: Multiple assignments

$x, y, z = 2, 5, 7$

$x, y = y, x$

$x = y = z = 7$

Appendix: Number-string conversion

```
str(float(int(float("7.5"))))
```

```
chr(sum(range(ord(min(str(not()))))))
```

Appendix: if-statements

```
if x % 2 == 0:  
    print('even')  
else:  
    print('odd')
```

```
if x < 0:  
    print('negative')  
elif x == 0:  
    print('zero')  
elif x == 1:  
    print('one')  
else:  
    print('>= 2')
```

Appendix: if-statements

Common mistake (if instead of elif)

```
if x == 0:  
    print('zero')  
if x % 2 == 0:  
    print('even')
```

Appendix: `while`-statements

```
x = 1
while x <= 5:
    print(x)
    x = x + 1
print("Done")
```

Appendix: None

If you can't say something nice,
then say None!

```
x = print(42)
print(x)  # prints None
```

```
if x is None:
    print("x is nothing")
else:
    print("x is something")
```

Appendix: or and and

```
x = 3
if x == 1:
    print("One")
elif x == 2 or x == 3:    # note, "or" not "and"!
    print("Some")
else:
    print("Many")
```

Appendix: String methods

```
s = 'this is a string'  
s.capitalize()  
s.upper()  
s.title()  
s.title().swapcase()  
s.removeprefix('this is '  
s.removesuffix(' string')  
s.replace('is', 'IS')  
s.count('i')
```

Appendix: Advice for chemists in lab

1. Keep track of units. One can easily be off by several orders of magnitude if there is a mistake with the units.
2. When working with long expressions divide it into smaller manageable parts.
3. Be careful with parentheses. In code $G/R*T$ means $(G/R)*T$ not $G/(R*T)$!
4. Collect reoccurring terms in a variable, e.g. $RT = R*T$ or $r = r_M + r_Q$.
This makes your code easier to read.
5. Use space in your code to make things easier to read.
 $r=r_M+r_Q$ is more difficult to read than $r = r_M + r_Q$.

— Author of `mod_i_lab`

Python for begyndere
`m.strova.dk/python`

Mathias Rav
`mathias@scalgo.com`

SCALGO



evaluering: <https://forms.gle/cHTjnrvtjrKDUqnw5>