Hvordan åbner jeg dette dokument?

Dette er en pdf-udskrift af iPython-Notebook-filen: `00-Handout.ipynb`. For at åbne selve iPython-filen kan du enten:

1. Med Google Colab

Gå ind på colab.research.google.com og upload filen under fanen "Upload".

2. Med Python installeret på din computer

Installér Python: Det gør du nemmest ved at installere den såkaldte Anaconda-distribution:

1. Gå ind på anaconda.com og tryk Download.
2. Følg installationsguiden
3. Tjek, at du har installeret korrekt ved at åbne programmet Anaconda-Navigator (læs mere her: docs.anaconda.org/anaconda/install/verify-install/)
4. Hvis du oplever problemer, kan du få mere hjælp her: https://docs.anaconda.com/anaconda/install/

Åbn filen:

1. Åbn Anaconda-Navigator fra dine programmer/apps på din computer.
2. Åbn **Jupyter Notebook** (med "Launch"-knappen)
3. Du ser nu en oversigt over dit filsystem. Navigér hen til den mappe, hvor filen ligger og dobbeltklik på den for at åbne den.

Python for begyndere

1. Hvad skal jeg skrive min Python-kode i?

Der er mange forskellige redigeringsværktøjer (IDE'er) til at skrive Python-kode: Bl.a. Google Colab, Jupyter Notebook, Spyder og Visual Studio Code. Jeg foretrækker selv Visual Studio, men som begynder vil jeg anbefale dig enten at bruge Colab (kræver ikke installering, men åbnes via Google Drive) eller Jupyter (køres "lokalt" på din computer, efter du har installeret Anaconda-distributionen (https://www.anaconda.com/) af Python). Både Colab og Jupyter kan åbne denne fil i et interaktivt "notebook"-format. Det betyder fx, at man kan skrive tekst som dette og vise grafer og resultatet af sine beregninger direkte i dokumentet, som jeg illustrerer nedenfor.

Genveje i Google Colab Notebook

Kommando	Genvej
Kør celle	shift + enter
Kør celle og indsæt ny under	alt (option) + enter
Ændr celle til tekst	ctr + M, M
Ændr celle til kode	ctr + M, Y
Fjern celle	ctr + D, D

Indtasting af specielle tegn

Tegn	Mac dansk tastatur	Windows dansk tastatur	US tastatur
{ }	alt + shift + 8 og alt + shift + 9	AltGR + 7 og Altgr + 0	Shift + [ ] (right to P key)
[ ]	alt + 8 og alt + 9	AltGR + 8 og Altgr + 9	[ ] (right to P key)
\	alt + shift + 7	ctrl + alt + < eller AltGr + 9	\ (left to enter)
_	shift + -	shift + -	shift + -

Tekst indskrives i markdown (læs mere her: https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet (https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet)).

Genveje i Jupyter Notebook (se kun hvis relevant)

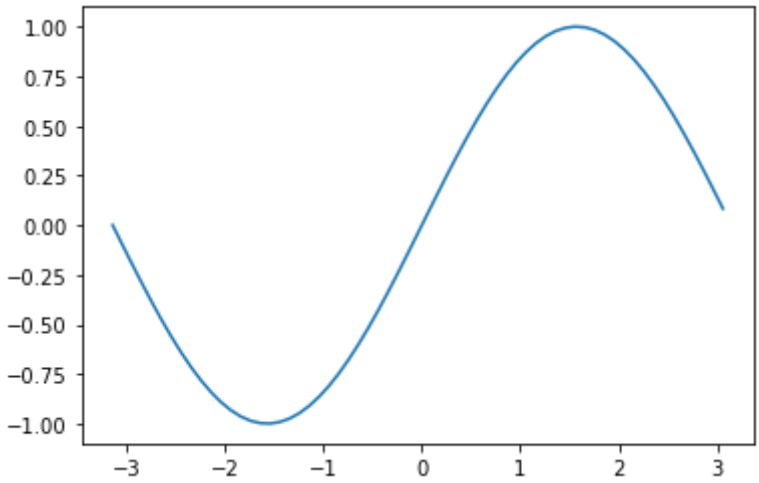
Kommando	Genvej
Kør celle	shift + enter
Kør celle og indsæt ny under	alt (option) + enter
Ændr celle til tekst	esc + M

Kommando	Genvej
Ændr celle til kode	esc + Y

Interaktive grafer

I en interaktiv notebook vises grafer og resultatet af beregninger direkte i output-feltet under python-kode-cellen:

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt

x = np.arange(-np.pi,np.pi,0.1)
y = np.sin(x)

plt.plot(x,y)
plt.show()
```



2. Kommandoer

Basale regnestykker

Plus, minus, dividere og gange udføres med normale tegn, som du kender det fra fx Maple eller Matlab.

In [ ]:
```
3+4-1
```

Out[4]: 6

In [ ]:
```
5/2
```

Out[5]: 2.5

In [ ]:
```
3*4
```

Out[6]: 12

Potenser klares med en dobbelt-asteriks $**$. Her fx 5^2

In [ ]:
```
5**2
```

Out[7]:
```
25
```

Python har indbyggede kommandoer. Fx til at afrunde og bestemme absolut værdi.

In [ ]:
```
round(3.4)
```

Out[8]:
```
3
```

In [ ]:
```
abs(-3)
```

Out[9]:
```
3
```

Print statements

Som udgangspunkt vises kun den sidste linje i en interaktiv Python Notebook som denne.

In [ ]:
```
3+4-1
5/2
```

Out[10]:
```
2.5
```

Ønsker man at se mellemregninger kan man altid skrive `print` efterfulgt af paranteser rundt om ens udtryk.

In [ ]:
```
print(3+4-1)
5/2
```

```
6
```

Out[11]:
```
2.5
```

Man kan også indsætte tekst og variable i paranteserne for at printe det.

In [ ]:
```
print("Hello World")
```

```
Hello World
```

In [ ]:
```
name = "Rune"
print("Hello " + name)
```

```
Hello Rune
```

3. Variabler og kommentarer

Brug lighedstegn for at definere en ny variabel. Fx som her, hvor vi vil holde styr på medlemmerne i en forening.

In [ ]:
```
member_count = 14
print(member_count)
```

```
14
```

Linjer der starter med `#` er kommentarer, som ikke bliver eksekveret. De forklarer og giver kontekst til koden.

In [ ]:
```
# The yearly fee collected at the General Assembly.
member_fee = 40
income = member_fee*member_count
print(income)
```

```
560
```

Man kan også lave kommentarer på flere linjer:

In [ ]:
```
'''
This is the management system for the new awesome
"Students who love Python" student assocation.
'''
```

Out[16]:
```
'\nThis is the management system for the new awesome\n"Students who lov
e Python" student assocation.\n'
```

Skriv tilpas (ikke for mange) kommentarer:

- Ingen kommentarer er foretrækket frem for forkerte eller outdatede kommentarer
- kommentarer bør ikke forklare hvad man indlysende kan læse direkte fra koden.

In [ ]:
```
member_count += 1
# member_count = member_count + 1
member_count
```

Out[17]:
```
15
```

4. Datatyper: Tal og tekst

Variabler kan repræsenterer mange ting. Når du definerer en variabel i Python, får den automatisk en datatype, som fortæller hvilken kategori, variablen tilhører. I princippet skal du ikke selv tænke over, hvad datatypen for din variabel er, men det er ofte nyttigt at vide det. Her er nogle af de vigtigste typer at kende:

Tal

`int` : Integer (heltal)

In [ ]:
```
member_count = 14
member_fee = 40
```

In [ ]:
```
x = 2
y = -2
x+y
```

Out[19]:
```
0
```

`float` : Floating point number (decimaltal)

Eksempel: Hvad er arealet af min pizza?

In [5]:
```
pizza_radius = 6.62
pizza_area = 3.14*(pizza_radius**2)
pizza_area
```

Out[5]:
```
137.608616
```

Eksempel: Hvad bør kontingentet være for at dække årlige bankudgifter på 699 kr.?

In [ ]:
```
min_fee = 699/member_count
min_fee
```

Out[21]:
```
49.92857142857143
```

`complex` : Complex float (komplekse tal)

In [ ]:
```
z = 2 + 3j
print(z)
```

```
(2+3j)
```

Tekst

`string` : String (tekst-streng)

In [6]:
```
name = 'Rune'
print('Hello ' + name + ' and welcome to the association!')
```

```
Hello Rune and welcome to the association!
```

For at indsætte tal i en tekst-streng, skal du eksplicit bede Python om at ændre dit tal til tekst med `str()` -kommandoen. Det kaldes mere generelt for type casting, når man konverterer mellem datatyper:

In [ ]:
```
print('We need your support of ' + str(round(min_fee)) + ' DKK to cover
```

```
We need your support of 50 DKK to cover our expenses
```

5. Datatyper: Lister

Lister skrives med hårde klammer `[]` og kan både indeholde tekst og tal.

In [ ]:
```
members = ["Emma", "Laura", "Magnus", "Rune"]
```

Lister i Python er "0-indekserede", så 1. element har indeks 0 (dvs. 0'te plads i listen):

In [ ]:
```
members[0]
```

Out[26]:
```
'Emma'
```

In [ ]:
```
members[2]
```

Out[27]:
```
'Magnus'
```

Udtræk 2. og 3. element (med indeks i det halvåbne interval $[1, 3[$)

In [ ]:
```
members[1:3]
```

Out[28]:
```
['Laura', 'Magnus']
```

Udtræk sidste element i listen:

In [ ]:
```
members[-1]
```

Out[29]:
```
'Rune'
```

Læg lister sammen og tæl antallet af elementer i listen.

In [ ]:
```
new_members = ["Luna", "Lukas"]
members + new_members
```

Out[30]:
```
['Emma', 'Laura', 'Magnus', 'Rune', 'Luna', 'Lukas']
```

In [ ]:
```
members # the list stays unchanged
```

Out[31]:
```
['Emma', 'Laura', 'Magnus', 'Rune']
```

In [ ]:
```
len(members)
```

Out[32]:
```
4
```

In [ ]:
```
member_fees = [0, 12.5, 34, 199.99]
```

OBS: Ganger du et tal med din liste, bliver listen blot kopieret.

In [ ]:
```
member_fees*2
```

Out[34]:
```
[0, 12.5, 34, 199.99, 0, 12.5, 34, 199.99]
```

Det intuitive havde nok været, at hvert element i listen blev ganget med tallet. For at få denne funktionalitet, skal man arbejde med NumPy Arrays frem for lister:

6. Datatyper: NumPy Arrays

NumPy er et bibliotek som bl.a. muliggør regning med vektorer og matricer, som du måske kender det fra Matlab. Det er essentielt til databehandling i Python.

In [ ]:
```
member_fees = [0, 12.5, 34, 199.99]
```

En NumPy Array er en datatype, der opfører sig som en vektor eller matrix. Du laver en ny array med `np.array()` -kommandoen:

In [ ]:
```
import numpy as np

x = np.array(member_fees)
x
```

Out[36]:
```
array([  0.  ,  12.5 ,  34.  , 199.99])
```

Hvis vi ganger vores array med 2, fungerer det som at gange en skalar på en vektor:

In [ ]:
```
x*2
```

Out[37]:
```
array([  0.  ,  25.  ,  68.  , 399.98])
```

NumPy har indbyggede kommandoer til at tilgå matematiske konstanter og elementære funktioner:

In [ ]:
```
sum(x*2)
```

Out[38]:
```
492.98
```

In [ ]:
```
pizza_radius = 6.62
```

In [ ]:
```
np.pi*pizza_radius**2
```

Out[40]:
```
137.67841308798054
```

Du kan også lave numerisk lineær algebra med NumPy

In [ ]:
```
y = np.array([[1, 2],
              [3, 4]])
y
```

Out[41]:
```
array([[1, 2],
       [3, 4]])
```

In [ ]:
```
z = np.arange(4).reshape(2,2)
z
```

Out[42]:
```
array([[0, 1],
       [2, 3]])
```

In [ ]:
```
np.dot(y,z)
```

Out[43]:
```
array([[ 4,  7],
       [ 8, 15]])
```

7. Booleans og betinget kode

En `boolean` er en værdi, som enten er sand eller falsk. Fx er det **sandt** at 3 er mindre end 4:

In [ ]:
```
3 < 4
```

Out[44]:
```
True
```

`booleans` bruges i betinget kode (`if statements`) til at styre flowet i ens program. Man skal give en betingelse som evaluerer til en sandt/falsk værdi og de kommandoer man ønsker opfyldt.

`if` -statements kan fx bruges til at beregne *cases* i matematik. Fx til at bestemme $f(-2)$ for følgende funktion:

$$f(x) = \begin{cases} x^3 + 2, & x > 0 \\ \frac{1}{x}, & \text{otherwise} \end{cases}$$

In [ ]:
```
x = -2
if x > 0:
  x = x**3 +2
else:
  x = 1/x
x
```

Out[45]:
```
-0.5
```

Det er indrykningen, der styrer hvad der kommer med i en `if` -statement:

In [8]:
```
member_fees = [0, 12.5, 34, 199.99]
expenses = 200
is_expenses_covered = expenses < sum(member_fees)
print(is_expenses_covered)
if (is_expenses_covered):
    print('Wuhuu! We made it.')
else:
    print('We need more money')
print('this line is printed no matter what')
```

```
True
Wuhuu! We made it.
this line is printed no matter what
```

Vælg et tilfældigt element fra en liste og kør noget kode afhængig af udfaldet:

In [ ]:
```
from random import choice

random_member = choice(["E", "L", "M", "R"])

print("who has to pay?")
if (random_member == "E"):
    print('Emma')
elif (random_member == "L"):
    print("Laura")
elif (random_member == "M"):
    print("Magnus")
else:
    print("Rune")
```

```
who has to pay?
Magnus
```

8. Funktioner

Funktioner er en måde at genbruge kodeblokke på, og begrebet minder en del om funktioner i matematik (men dog med forskelle). En funktion er en maskine, der

1. (valgfrit) tager et input ind (parameter/argument).
2. Udfører nogle beregninger og kører nogle kommandoer
3. (valgfrit) spytter et resultat ud (det kalder vi at "returnere" en variabel).

Vi skal lige have lidt mere terminiologi på plads: Når vi bruger en funktion, siger vi at vi "kalder" den (eng. "call a function"). Vi benævner også ofte en funktion som en "metode" (eng. "method") eller "kommando" (eng. "command"). I Python kalder man en funktion ved at skrive dens navn efterfulgt af paranteser, og det har vi faktisk allerede brugt flere gange tidligere i dokumentet. Fx:

In [12]:
```
print("Hello world")
```

```
Hello world
```

In [13]:
```
round(3.14)
```

Out[13]:
```
3
```

Vi kan selv definere en funktion. Fx en der hedder `add_member`, som tilføjer et navn til listen `members`. Bemærk, at funktionen selv bruger en funktion, nemlig `append`, som kaldes direkte på listen `members`.

In [10]:
```
members = ["Emma", "Laura", "Magnus", "Rune"]
```

In [11]:
```
def add_member(name):
    '''Adds the name to the members list.
    '''
    members.append(name)

add_member("Carl")
print(members)
```

```
['Emma', 'Laura', 'Magnus', 'Rune', 'Carl']
```

Vi kan fjerne navnet igen ved at "kalde" `remove()` -funktionen på listen:

In [14]:
```
members.remove('Carl')
members
```

Out[14]:
```
['Emma', 'Laura', 'Magnus', 'Rune']
```

Vi kan modificere funktionen, så det samme navn ikke kan blive tilføjet 2 gange:

In [15]:
```
def add_member(name):
    '''Adds the name to the members list.
    '''
    if name not in members:
        members.append(name)

add_member("Carl")
print(members)
```

```
['Emma', 'Laura', 'Magnus', 'Rune', 'Carl']
```

Indrykning

Ligesom med `if` -statements er det indrykningen, der afgør hvilke linjer, der bliver kørt.

In [ ]:
```
def print_lines():
    print('Line 1')
    print('Line 2')

print_lines()
```

```
Line 1
Line 2
```

In [ ]:
```
def print_lines():
    print('Line 1')
print('Line 2')

print_lines()
```

```
Line 2
Line 1
```

Karakterberegner eksempel

Forestil dig, at du skal på udveksling og derfor har brug for at omregne dit karaktersnit fra

kurser i Danmark til den internationale ects-skala.

Vi kan lave vores egen funktion til at beregne gennemsnit:

In [ ]:
```
grades = [10, 2, 7]

def grades_mean(grades):
    '''Return mean of grades list.'''
    return sum(grades)/len(grades)

gpa = grades_mean(grades)
print(gpa)
```

```
6.333333333333333
```

Eller vi kan bruge NumPy's indbyggede funktion til at beregne gennemsnit:

In [ ]:
```
import numpy as np

grades = [10, 2, 7]

gpa = np.mean(grades)
print(gpa)
```

```
6.333333333333333
```

Værtsuniversitetet ønsker dine karakterer på ECTS-skalaen, så du laver din egen funktion til at konvertere karakterer fra den danske til internationale skala:

In [ ]:
```
def convert_grade(dk_grade):
    dk_scale = [-3, 0, 2, 4, 7, 10, 12]
    ects_scale = ["F", "Fx", "E", "D", "C", "B", "A"]
    index = dk_scale.index(dk_grade)
    return ects_scale[index]

print(convert_grade(7))
```

```
C
```

9. Løkker og list comprehensions

Løkker

Ønsker man at gentage en linje kode flere gange skal man bruge et `for loop` (en løkke på dansk). Fx til at gange alle elementer i en liste

In [ ]:
```
for i in [0, 1, 2]:
    print(i)
```

```
0
1
2
```

In [ ]:
```
for i in [0, 1, 2]:
    print(i*2)
```

```
0
2
4
```

Konvertér samtlige karakterer på én gang:

In [ ]:
```
for grade in grades:
    print(convert_grade(grade))
```

```
B
E
C
```

Omregn grader Celcius til Kelvin for 3 temperaturmålinger:

In [ ]:
```
temperatures = [25.5, 22.2, 38]

for T in temperatures:
    print(str(T + 273.15) + ' K')
```

```
298.65 K
295.34999999999997 K
311.15 K
```

List comprehensions

List comprehensions er et shorthand alternativ til at lave for loops:

In [ ]:
```
[T + 273.15 for T in temperatures]
```

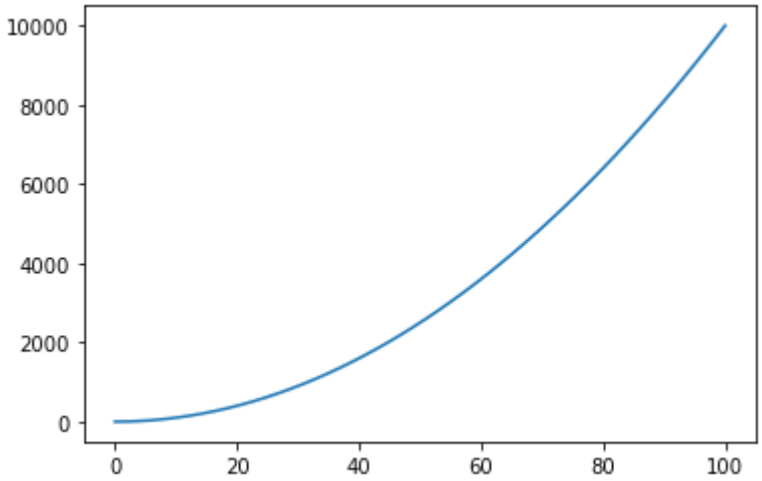
Out[63]:
```
[298.65, 295.34999999999997, 311.15]
```

Flere Eksempler

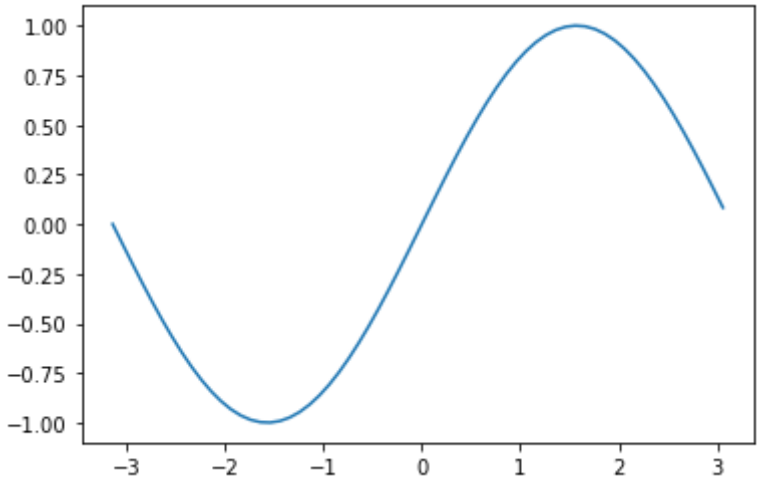
Her er flere eksempler på brug af Python. Jeg har ladet beskrivelserne være på engelsk.

Example 1 - Graph of mathematical function

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(0,100)
y = x**2
plt.plot(x,y)
plt.show()
```



In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt

x = np.arange(-np.pi,np.pi,0.1)
y = np.sin(x)

plt.plot(x,y)
plt.show()
```



More functions in same plot

In [ ]:
```
z = np.cos(x)

plt.plot(x,y)
plt.plot(x,z)
plt.show()
```

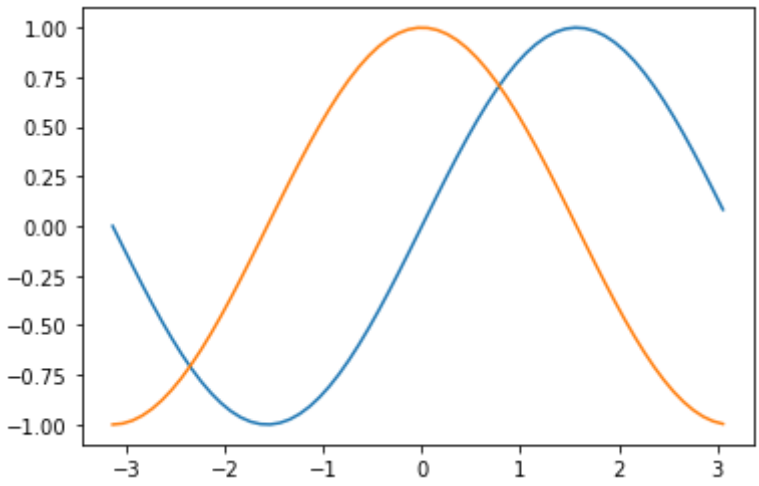
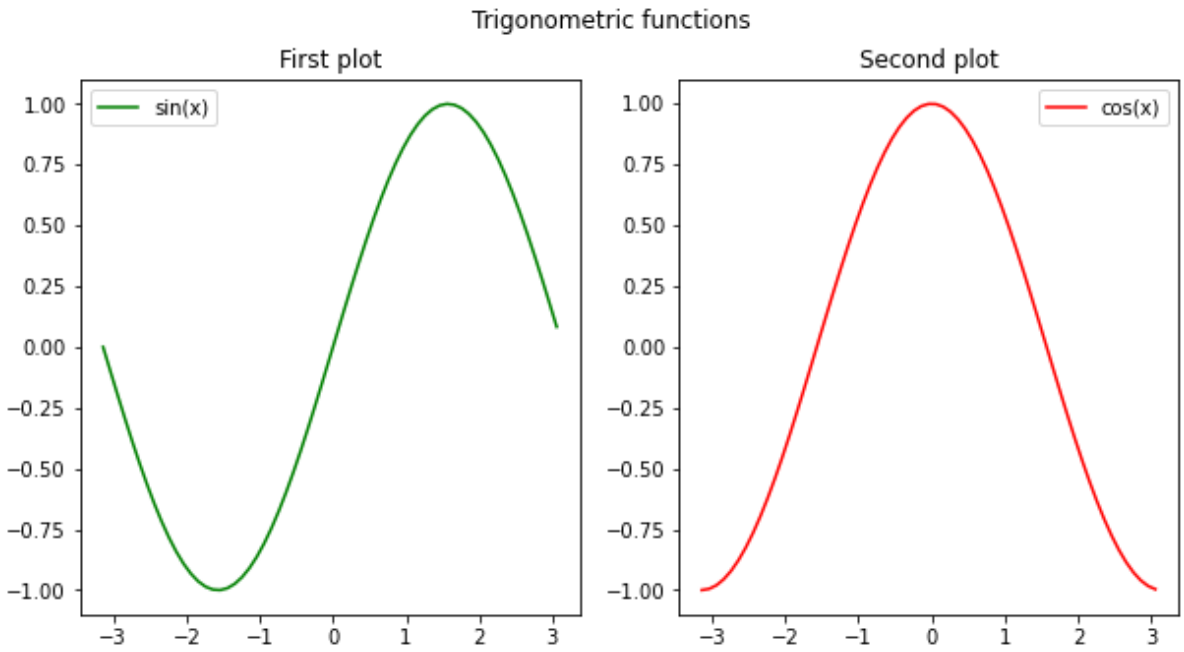


Figure with subplots

In [ ]:
```
fig, axs = plt.subplots(1, 2, figsize=(10,5))
fig.suptitle('Trigonometric functions')
axs[0].plot(x,y,'g',label='sin(x)')
axs[0].legend()
axs[0].set_title('First plot')

axs[1].plot(x,z,'r',label='cos(x)')
axs[1].legend()
axs[1].set_title('Second plot')

plt.show()
```



Export figure/plot to pdf:

In [ ]:
```
plt.plot(x,y)
plt.savefig('./sinx.pdf', bbox_inches='tight') # Save figure in current
plt.close() # Close current figure to hide it from the output.
```

Example 2 - Read data from csv

In [ ]:
```
import numpy as np
import pandas as pd
data = pd.read_csv('data.csv')
x = np.array(data['x'])
y = np.array(data['y'])
x,y
```

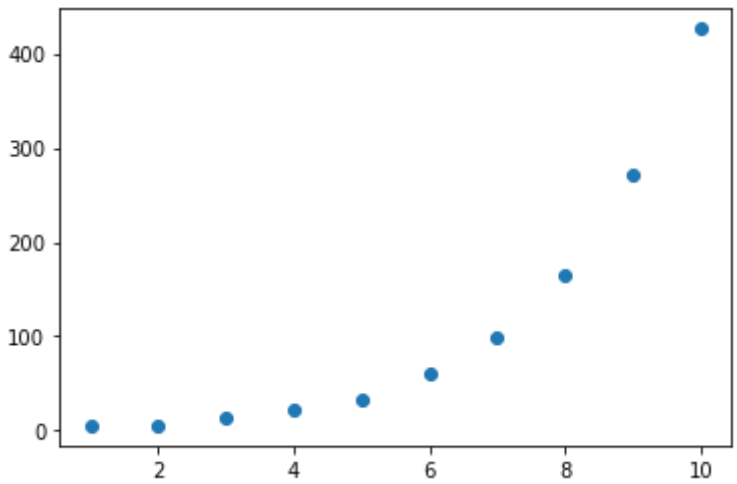
Out[69]:
```
(array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10]),
 array([  5.44,   6.13,  13.11,  22.16,  32.33,  61.39,  98.27, 164.02,
        270.38, 426.18]))
```

In [ ]:
```
data
```

Out[70]:

	x	y
0	1	5.44
1	2	6.13
2	3	13.11
3	4	22.16
4	5	32.33
5	6	61.39
6	7	98.27
7	8	164.02
8	9	270.38
9	10	426.18

In [ ]:
```
plt.plot(x,y,'o')
plt.show()
```



Example 3 - Regression on data

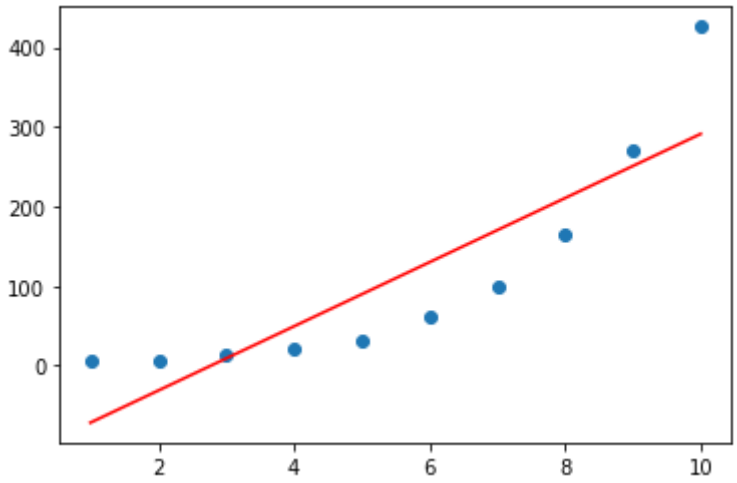
In [ ]:
```
import sklearn.linear_model as lm

def linear_regression(x, y):
    X = np.array(x).reshape(-1,1)
    y = np.array(y)

    # Fit ordinary least squares regression model
    model = lm.LinearRegression(fit_intercept=True)
    model = model.fit(X,y)

    score = model.score(X, y)
    a = model.coef_
    b = model.intercept_
    return model, score, a, b
```

In [ ]:
```
X = np.arange(min(x)-0.01,max(x)+0.01,0.01).reshape(-1,1)
```

In [ ]:
```
model, score, a, b = linear_regression(x, y)
Y_est = model.predict(X)
```

Linear regression

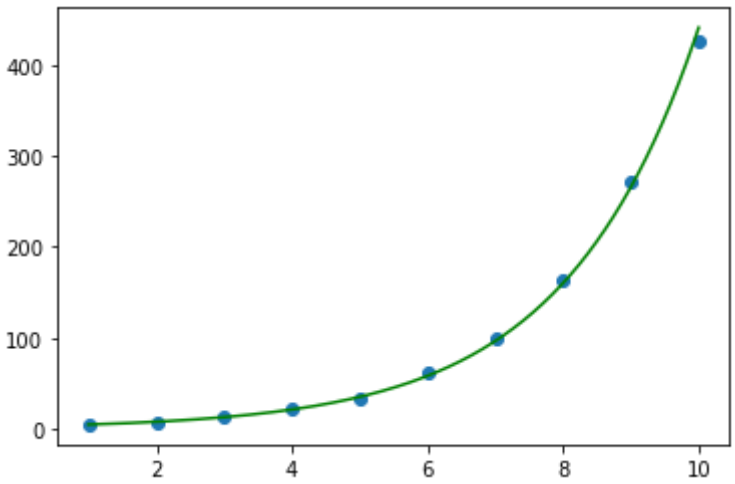
In [ ]:
```
plt.plot(x,y,'o')
plt.plot(X,Y_est,'-r')
# plt.plot(X,a*X+b,'-g') # Alternatively
plt.show()
```



Exponential regression

In [ ]:
```
exp_model, score, a, b = linear_regression(x, np.log10(y))
Z_est = exp_model.predict(X)
print(score, a, b)
plt.plot(x,y,'o')
plt.plot(X,10**Z_est,'-g')
# plt.plot(X,(10**b)*(10**a)**X,'-g') # Alternatively
plt.show()
```

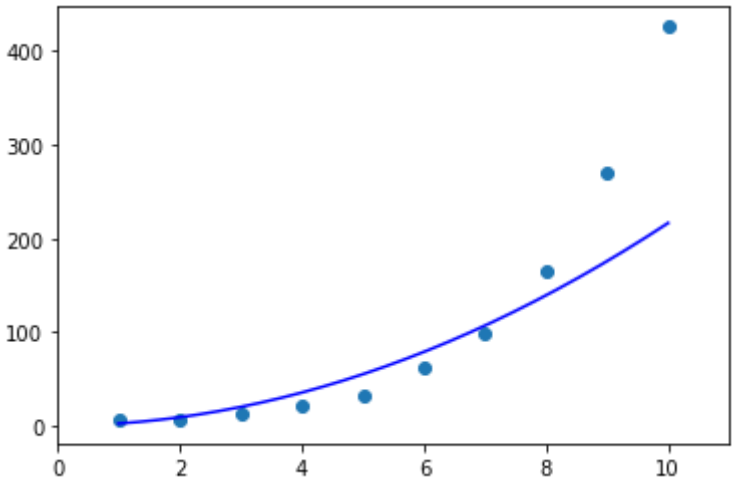
```
0.995677236894319 [0.21977613] 0.44651331454462984
```



Power Regression

In [ ]:
```
power_model, score, a, b = linear_regression(np.log10(x), np.log10(y))
W_est = power_model.predict(np.log10(X))
print(score, a, b)
plt.plot(x,y,'o')
plt.plot(X,10**W_est,'-b')
# plt.plot(X,(10**b)*(X**a),'-g') # Alternatively
plt.xlim(0,11)
plt.show()
```

```
0.8905727844760719 [1.97678413] 0.3585585117920571
```



Example 4 - Working with files

In [ ]:
```
import os
print([dirs for root, dirs, files in os.walk("./")])
```

```
[['.ipynb_checkpoints'], []]
```

In [ ]:
```
import os

files = [files for root, dirs, files in os.walk("./")]

print(files)
```

```
[['00-Python for begyndere-2022-04-19.ipynb', '.DS_Store', 'hubble.jpg
', 'data.csv', 'Pandas_Cheat_Sheet.pdf', 'Numpy_Python_Cheat_Sheet.pdf
', 'sinx.pdf'], ['Python for begyndere-2022-02-22-checkpoint.ipynb', '0
0-Python for begyndere-checkpoint.ipynb', '00-Python for begyndere-2022
-04-19-checkpoint.ipynb', 'Python for begyndere-checkpoint.ipynb', '00-
Python for begyndere-2022-02-22-checkpoint.ipynb']]
```

In [ ]:
```
f = open('./data.csv')
lines = f.readlines()
for line in lines[1:]:
    print(line[:-2].split(','))
```

```
['1', '05.4']
['2', '06.1']
['3', '13.1']
['4', '22.1']
['5', '32.3']
['6', '61.3']
['7', '98.2']
['8', '164.0']
['9', '270.3']
['10', '426.']
```

Example 5 - Interfaces

Simple command line interfaces er nemme at lave i Python:

In [ ]:
```
str = input("Would you like some ice cream? (y/n): ")
print("Received input is: ", str)
7
if (str == 'y'):
    print("You get an ice cream!")
elif (str == 'n'):
    print("No ice cream sorry.")
else:
    print("Please choose either y or n")
```

```
Would you like some ice cream? (y/n): y
Received input is:  y
You get an ice cream!
```

GUIs and graphical web applications

Python is in my personal opinion not really suited for building Graphical User Interfaces (GUIs).

If you wish to build a profesional GUI I would recommend a web solution - for bigger frontends (interfaces, which the user interact with directly) you can for instance use React with javascript and then Python with Django for building a server-backend (functionality, which the user doesn't interact with directly). However this is a professional solution which require some level of software engineering skills. As an alternative there does exist some Python packages, which you may use, if you insist on building a GUI with Python:

1. tkinter (part of Python standard library): https://docs.python.org/3/library/tkinter.html (https://docs.python.org/3/library/tkinter.html)
2. PyQt (installed with Anaconda): https://wiki.python.org/moin/PyQt (https://wiki.python.org/moin/PyQt).

Off course, these packages also require coding skills beyond beginner level to use.

Example 6 - Load image data

In [ ]:
```
import numpy as np
from PIL import Image

img = Image.open('hubble.jpg')
img_array = np.array(img)
print(img_array.shape)
# img.show()
```

```
(1200, 1200, 3)
```

Count white pixels fraction

In [ ]:
```
# Count pixel values.
img_array[img_array > 254].size / img_array.size
```

Out[83]:
```
0.004540046296296296
```

Example 7 - Managing dependencies with Conda Environments

The following is slightly more advanced than what is needed for the first introduction, but I've included it, since it might come in handy. When you are many people working on the same project, you need to ensure, that all have the same packages and versions installed. You can manage this with conda environments.

1. Open up a terminal and navigate to the project directory (`cd path/to/project`). Create environment (only first time) with specified Python version by running the following:

   ```
   conda create –n myProject python=3.7.6
   ```

2. Activate environment:

   ```
   conda activate myProject
   ```

3. Make text-file `requirements.txt` listing all requirements. The contents of the file could e.g. be:

```
numpy
matplotlib
pandas
```

4. Install all requirements with pip

```
pip install -r requirements.txt
```