

PYTHON 2 (LET ØVET)

FOR IDA D. 26/03/2023

Omkring mig

- Anders Bensen Ottsen
 - ”Underviser” i dag
- Diplomingeniør i Softwareteknologi
 - Fra Syddansk Universitet
- (Næsten!) Civilingeniør i Computer Science & Engineering
 - Fra Danmarks Tekniske Universitet
- Arbejder pt. som Machine Learning Engineer ved Weel & Sandvig
 - Starter som Data Scientist ved PFA her 1. April
- Det her er 5. gang jeg underviser ”Python 2” for IDA

Omkring Jens

- Jens Kristian Vitus Bering
 - Fra IDA & hjælper med opgaver
- BSc. I Software Engineering
 - Fra Syddansk Universitet
- Læser cand.polyt i Software Engineering
 - Ved Syddansk Universitet
- Arbejder som studenterudvikler ved Bankdata
 - Hvor han programmerer finans software

Dagens program

- Klasser
- Filhåndtering
- Biblioteker i Python
- Grafisk brugergrænseflade
- Avanceret emner
- Hvad man kan gøre herfra og spørgerunde

Dagens program

- Klasser

Objektorienteret programmering

- Filhåndtering
- Biblioteker i Python
- Grafisk brugergrænseflade
- Avanceret emner
- Hvad man kan gøre herfra og spørgerunde

Dagens program

- Klasser

Objektorienteret programmering

- Filhåndtering

- Biblioteker i Python

- Grafisk brugergrænseflade

Mere avanceret programmering

- Avanceret emner

- Hvad man kan gøre herfra og spørgerunde

Dagens program

- Klasser

Objektorienteret programmering

- Filhåndtering

- Biblioteker i Python

Mere avanceret programmering

- Grafisk brugergrænseflade

- Avanceret emner

Machine Learning

- Hvad man kan gøre herfra og spørgerunde

Dagens program

- Klasser

Objektorienteret programmering

- Filhåndtering

- Biblioteker i Python

Mere avanceret programmering

- Grafisk brugergrænseflade

- Avanceret emner

Machine Learning

- Hvad man kan gøre herfra og spørgerunde

Afrunding

Dagens program

- Klasser

Objektorienteret programmering

- Filhåndtering

- Biblioteker i Python

Mere avanceret programmering

- Grafisk brugergrænseflade

- Avanceret emner

Machine Learning

- Hvad man kan gøre herfra og spørgerunde

Afrunding

Meget abstrakt

Dagens program

Vigtigst i dag

- Klasser

Objektorienteret programmering

- Filhåndtering

- Biblioteker i Python

- Grafisk brugergrænseflade

Mere avanceret programmering

- Avanceret emner

Machine Learning

- Hvad man kan gøre herfra og spørgerunde

Afrunding

Meget abstrakt

Efter i dag kan I:

- Forstå objektorienteret programmerings koncepter
- Se nyttigheden i biblioteker og selv installere og bruge dem
- Formulere og forstå mere avanceret Python programmer
- Så hvordan når vi der til?
 - Ved små ”forelæsninger”
 - Med lidt liveprogrammering fra undertegnet
 - Opgaveløsning

Efter i dag kan I:

- Forstå objektorienteret programmerings koncepter
- Se nyttigheden i biblioteker og selv installere og bruge dem
- Formulere og forstå mere avanceret Python programmer
- Så hvordan når vi der til?
 - Ved små ”forelæsninger”
 - Med lidt liveprogrammering fra undertegnet

- Opgaveløsning

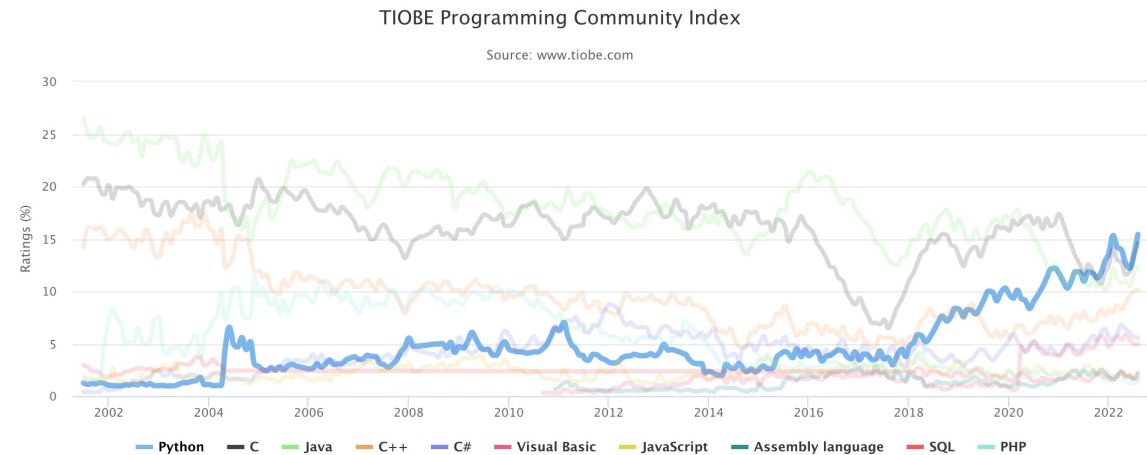
**Det meste af
tiden går her**

Det online format

- Forelæsningerne bliver mig der taler i zoom
 - Bare unmute jer og tal løs hvis der er spørgsmål
- Ved opgaverne bliver I smidt ud i breakout rooms
 - Hvor I har ca. 20-30 minutter til at løse dem
 - Jens, Loc og jeg hopper rundt og hjælper i breakout rooms
 - ”Ræk hånden op” hvis i sidder fast! Så kommer vi (prøver i hvert fald)
 - I er selvfølgelig ikke tvunget til at tale med de andre, men det plejer at hjælpe!
- PPT
 - TODO
- PDF:
 - TODO

Python

- Et multi-paradigme programmeringssprog
 - Paradigme → en speciel måde at anskue tingene på
- Opfundet i 1991
- Et af verdens mest populære programmeringssprog



Visual Studio Code

- IDE'en vi bruger i dag
- Minimalistisk
- Virkelig mange gratis tilføjelsespakker
- Er generelt virkelig god!

Python recap

- Hurtigt live eksempel
 - Variabler, funktioner, syntax, eksekvering

Klasser

Objektorienteret Programmering

- I kender det at skrive et lille Python program, hvor vi bare har en fil der klarer alt
 - Umuligt at vedligeholde
 - Hvorfor ikke ”modellere” sin kode til sit ansvarsområde? → OOP
- Python er jo som sagt multi-paradigme, og understøtter her i blandt OOP
- Vi prøver at modellere verden’ i koden
 - Ret abstrakt

Klasser

- I OOP bruger man klasser til at beskrive et abstrakt stykke af verden
- Den beskriver en slags "type" af et objekt
- Eksempler på klasser:
 - Bord, stol, bil, patient, person, øl

Klasser

- Er kendetegnet ved 3 ting:
 - Variabler (attributter)
 - Bruges til at beskrive en klasse
 - Kender vi
 - Metoder
 - Bruges til at beskrive hvad en klasse kan
 - Kender vi
 - Constructor
 - Hvad er det?

Constructor

- En slags metode
- Bruges til at konstruere en klasse
 - Deraf construct...
- Man fodrer den tingene der kendetegner en klasse

Klassen bord

- Hvad kendetegner et bord?
 - Antal ben
 - Materiale
 - Højde
 - Osv.

Et bord i Python

```
1  class Table:
2      amount_of_legs = 0
3      material = ""
4      height = 0
5
6      def __init__(self, a_o_l, m, h):
7          self.amount_of_legs = a_o_l
8          self.material = m
9          self.height = h
10
11     def print_table(self):
12         print(str(self.amount_of_legs) + " " + self.material + " " + str(self.height))
```

Et bord i Python

```
1  class Table:
2      amount_of_legs = 0
3      material = ""
4      height = 0
5
6      def __init__(self, a_o_l, m, h):
7          self.amount_of_legs = a_o_l
8          self.material = m
9          self.height = h
10
11     def print_table(self):
12         print(str(self.amount_of_legs) + " " + self.material + " " + str(self.height))
```

Attributter

Et bord i Python

```
1  class Table:
2      amount_of_legs = 0
3      material = ""
4      height = 0
5
6      def __init__(self, a_o_l, m, h):
7          self.amount_of_legs = a_o_l
8          self.material = m
9          self.height = h
10
11     def print_table(self):
12         print(str(self.amount_of_legs) + " " + self.material + " " + str(self.height))
```

Attributter

Constructor

Et bord i Python

```
1  class Table:
2      amount_of_legs = 0
3      material = ""
4      height = 0
5
6      def __init__(self, a_o_l, m, h):
7          self.amount_of_legs = a_o_l
8          self.material = m
9          self.height = h
10
11     def print_table(self):
12         print(str(self.amount_of_legs) + " " + self.material + " " + str(self.height))
```

Attributter

Constructor

Metode

The diagram illustrates the components of a Python class named 'Table'. It consists of three parts: 1. Attributes: Lines 2-4 show class-level attributes 'amount_of_legs' (set to 0), 'material' (set to an empty string), and 'height' (set to 0). These are circled in purple and labeled 'Attributter'. 2. Constructor: Lines 6-9 show the '__init__' method, which initializes instance variables 'self.amount_of_legs', 'self.material', and 'self.height' based on the arguments 'a_o_l', 'm', and 'h'. This is circled in purple and labeled 'Constructor'. 3. Method: Lines 11-12 show the 'print_table' method, which prints the values of the instance variables. This is circled in purple and labeled 'Metode'. A large purple oval encompasses the entire class definition.

Objekter

- Er en instans af en klasse
- F.eks. Finder der jo flere forskellige typer borde
 - Spiseborde, sofaborde, skriveborde ...

Objekter



Objekter i Python

```
1  from table import Table
2
3  kitchen_table = Table(4, "Wood", 120)
4  living_room_table = Table(4, "Plastic", 60)
5  office_table = Table(2, "Metal", 130)
6
7  kitchen_table.print_table()
8  living_room_table.print_table()
9  office_table.print_table()
```

Objekter i Python

```
1 from table import Table
2
3 kitchen_table = Table(4, "Wood", 120)
4 living_room_table = Table(4, "Plastic", 60)
5 office_table = Table(2, "Metal", 130)
6
7 kitchen_table.print_table()
8 living_room_table.print_table()
9 office_table.print_table()
```

Vi importerer Table klassen fra vores fil table.py

Objekter i Python

```
1 from table import Table
2
3 kitchen_table = Table(4, "Wood", 120)
4 living_room_table = Table(4, "Plastic", 60)
5 office_table = Table(2, "Metal", 130)
6
7 kitchen_table.print_table()
8 living_room_table.print_table()
9 office_table.print_table()
```

Vi importerer Table klassen fra vores fil table.py

Vi instantierer vores bord objekter

Objekter i Python

```
1 from table import Table
```

Vi importerer Table klassen fra vores fil table.py

```
2
```

```
3 kitchen_table = Table(4, "Wood", 120)
```

```
4 living_room_table = Table(4, "Plastic", 60)
```

Vi instantierer vores bord objekter

```
5 office_table = Table(2, "Metal", 130)
```

```
6
```

```
7 kitchen_table.print_table()
```

```
8 living_room_table.print_table()
```

Vi kalder objektets metode

```
9 office_table.print_table()
```


Objekter i Python

```
1 from table import Table
```

Vi importerer Table klassen fra vores fil table.py

```
2
```

```
3 kitchen_table = Table(4, "Wood", 120)
```

```
4 living_room_table = Table(4, "Plastic", 60)
```

Vi instantierer vores bord objekter

```
5 office_table = Table(2, "Metal", 130)
```

```
6
```

```
7 kitchen_table.print_table()
```

```
8 living_room_table.print_table()
```

Vi kalder objektets metode

```
9 office_table.print_table()
```

- Live!

Opgaver pt. 1 (20 minutter)

- 1. Lav en klasse som repræsentere en øl (Kald den Beer, ikke øl). En øl består af de tre attributter: navn(name), alkoholprocent(percentage) og et mærke(brand). Derudover består en øl af en metode: def print_beer() til at printe de tre ting der kendetegner en øl.**
 - Når klassen & metoden er lavet, så lav en ny fil, importer den fil du lavede dine klasser, og brug klassen til at lave 3 objekter der repræsentere dine yndlingsøl.
- 2. (Ekstra) Lav en metode som kan ændre på alkoholprocenten efter du har "instantieret" objektet, metoden skal tage den nye procent som argument**
 - Hint:
 - Det minder ret meget om constructor metoden

Filhåndtering

Filer

- I computere bruger vi filer til at opbevare noget data eller et program
 - Alle de programmer vi har lavet er f.eks. bare filer med noget specifikt kode
- Vi kan også bruge filer til at gemme data, som vi så kan bruge inde i vores program!
 - Så inde i et Python program bruger vi en anden fil til at styre data

Filer

- Ofte bruger man ".txt" filer til at opbevare data
 - Du kan så enten skrive nyt data til den her fil, eller læse gammel data
- Filer kan altså bruges som en slags "database"
- Forestil jer f.eks. et scenarie hvor vi har et kalender program
 - Når vi skriver data til programmet, så gemmer det det også i filen
 - Fordi så når vi åbner programmet næste gang, kan den læse alt dataen ind fra filen!

Eksempel fil

≡ couples.txt ×

examples > ≡ couples.txt

```
1 Jens,Anne
2 Anders,Rose
3 Hans,Birgitte
4 Peter,Line
5 Bastian,Freja
```

- *couples.txt* består af par og deres navne
- Der er 5 par
 - Hvor mandens navn er til venstre for kommaet og kvindens navn er til højre for kommaet
 - Så vi bruger altså et komma til at separere data!

Skrive til en fil i Python

```
1 my_file = open("my_file.txt", "w")  
2  
3 my_file.write("Jens")  
4  
5 my_file.close()
```

Skrive til en fil i Python

```
1 my_file = open("my_file.txt", "w")
2
3 my_file.write("Jens")
4
5 my_file.close()
```

Vi bruger ***open*** metoden til at åbne filen ***my_file.txt*** og fortæller at vi gerne vil skrive til filen (***w***)

open metoden returnerer et fil
objekt fra klassen File vi kan bruge til
at læse/skrive til en fil

Skrive til en fil i Python

```
1 my_file = open("my_file.txt", "w")
2
3 my_file.write("Jens")
4
5 my_file.close()
```

Vi bruger ***open*** metoden til at åbne
filen ***my_file.txt*** og fortæller at vi
gerne vil skrive til filen (***w***)

open metoden returnerer et fil objekt fra klassen File vi kan bruge til at læse/skrive til en fil

Skrive til en fil i Python

```
1 my_file = open("my_file.txt", "w")
2
3 my_file.write("Jens")
4
5 my_file.close()
```

Vi bruger ***open*** metoden til at åbne filen ***my_file.txt*** og fortæller at vi gerne vil skrive til filen (***w***)

Vi bruger ***write*** metoden til at skrive strengen "Jens" til vores fil

open metoden returnerer et fil objekt fra klassen File vi kan bruge til at læse/skrive til en fil

Skrive til en fil i Python

```
1 my_file = open("my_file.txt", "w")
2
3 my_file.write("Jens")
4
5 my_file.close()
```

Vi bruger **open** metoden til at åbne filen **my_file.txt** og fortæller at vi gerne vil skrive til filen (**w**)

Vi bruger **write** metoden til at skrive strengen "Jens" til vores fil

Til sidst bruger vi **close** metoden til at lukke vores fil. Hvis ikke vi gør dette så kan vi overbelaste vores computer!

open metoden returnerer et fil objekt fra klassen File vi kan bruge til at læse/skrive til en fil

Skrive til en fil i Python

```
1 my_file = open("my_file.txt", "w")
2
3 my_file.write("Jens")
4
5 my_file.close()
```

Vi bruger ***open*** metoden til at åbne filen ***my_file.txt*** og fortæller at vi gerne vil skrive til filen (***w***)

Vi bruger ***write*** metoden til at skrive strengen "Jens" til vores fil

Til sidst bruger vi ***close*** metoden til at lukke vores fil. Hvis ikke vi gør dette så kan vi overbelaste vores computer!

- Live!

Skrive en liste til en fil i Python

```
1 my_file = open("name_file.txt", "w")
2 names = ["Jens", "Anders"]
3 for n in names:
4     my_file.write(n + "\n")
5 my_file.close()
```

Skrive en liste til en fil i Python

```
1 my_file = open("name_file.txt", "w")
2 names = ["Jens", "Anders"]
3 for n in names:
4     my_file.write(n + "\n")
5 my_file.close()
```

Meget vigtigt at tilføje "**n**" til sidst, fordi ellers kommer der ikke en ny linje inde i filen!

Skrive en liste til en fil i Python

```
1 my_file = open("name_file.txt", "w")
2 names = ["Jens", "Anders"]
3 for n in names:
4     my_file.write(n + "\n")
5 my_file.close()
```

Meget vigtigt at tilføje "**n**" til sidst, fordi ellers kommer der ikke en ny linje inde i filen!

- Live!

Læse fra en fil i Python

```
1  my_file = open("name_file.txt","r")
2
3  first_line = my_file.readline() # "Jens"
4  all_lines = my_file.readlines() # ["Jens", "Anders"]
5
6  my_file.close()
```


Læse fra en fil i Python

```
1  my_file = open("name_file.txt", "r")
2
3  first_line = my_file.readline() # "Jens"
4  all_lines = my_file.readlines() # ["Jens", "Anders"]
5
6  my_file.close()
```

Vi vil gerne læse fra filen
(*r*)

Læse fra en fil i Python

```
1  my_file = open("name_file.txt", "r")
2
3  first_line = my_file.readline() # "Jens"
4  all_lines = my_file.readlines() # ["Jens", "Anders"]
5
6  my_file.close()
```

Læs en linje

Vi vil gerne læse fra filen
(*r*)

Læse fra en fil i Python

```
1 my_file = open("name_file.txt", "r")
2
3 first_line = my_file.readline() # "Jens"
4 all_lines = my_file.readlines() # ["Jens", "Anders"]
5
6 my_file.close()
```

Læs en linje

Læs alle linjer

Vi vil gerne læse fra filen
(*r*)

Læse fra en fil i Python

```
1  my_file = open("name_file.txt", "r")
2
3  first_line = my_file.readline() # "Jens"
4  all_lines = my_file.readlines() # ["Jens", "Anders"]
5
6  my_file.close()
```

Læs en linje

Læs alle linjer

Vi vil gerne læse fra filen
(*r*)

- Live!

Læse *couples.txt* filen ind i Python

```
1  males = []
2  females = []
3
4  with open('couples.txt') as f:
5      for line in f:
6          couples = line.split(",")
7          males.append(couples[0])
8          females.append(couples[1].rstrip())
9
```

Læse *couples.txt* filen ind i Python

```
1  males = []
2  females = []
3
4  with open('couples.txt') as f:
5      for line in f:
6          couples = line.split(",")
7          males.append(couples[0])
8          females.append(couples[1].rstrip())
9
```

Med keywordet ***with*** kalder vi det returnerede objekt fra ***open*** metoden ***f***

Læse *couples.txt* filen ind i Python

```
1  males = []
2  females = []
3
4  with open('couples.txt') as f:
5      for line in f:
6          couples = line.split(",")
7          males.append(couples[0])
8          females.append(couples[1].rstrip())
9
```

Med keywordet ***with*** kalder vi det returnerede objekt fra ***open*** metoden ***f***

Vi løber igennem hver linje i filen ***f***

Læse *couples.txt* filen ind i Python

```
1  males = []
2  females = []
3
4  with open('couples.txt') as f:
5      for line in f:
6          couples = line.split(",")
7          males.append(couples[0])
8          females.append(couples[1].rstrip())
9
```

Med keywordet ***with*** kalder vi det returnerede objekt fra ***open*** metoden ***f***

Vi løber igennem hver linje i filen ***f***

Vi siger at på hver linje i filen vi får, skal vi splitte linjen op ved kommaet

Læse *couples.txt* filen ind i Python

```
1  males = []
2  females = []
3
4  with open('couples.txt') as f:
5      for line in f:
6          couples = line.split(",")
7          males.append(couples[0])
8          females.append(couples[1].rstrip())
9
```

Med keywordet **with** kalder vi det returnerede objekt fra **open** metoden **f**

Vi løber igennem hver linje i filen **f**

Vi siger at på hver linje i filen vi får, skal vi splitte linjen op ved kommaet

Couples er nu en liste, hvor vi ved at venstre side af kommaet er mand

Læse *couples.txt* filen ind i Python

```
1  males = []
2  females = []
3
4  with open('couples.txt') as f:
5      for line in f:
6          couples = line.split(",")
7          males.append(couples[0])
8          females.append(couples[1].rstrip())
9
```

Med keywordet **with** kalder vi det returnerede objekt fra **open** metoden **f**

Vi løber igennem hver linje i filen **f**

Vi siger at på hver linje i filen vi får, skal vi splitte linjen op ved kommaet

Couples er nu en liste, hvor vi ved at venstre side af kommaet er mand

Det sidste element der er i listen, bruger vi metoden **rstrip** til at fjerne støj, fordi i filer bliver der tilføjet "**\n**" til slut i linjen for at skifte til næste linje

Læse *couples.txt* filen ind i Python

```
1  males = []
2  females = []
3
4  with open('couples.txt') as f:
5      for line in f:
6          couples = line.split(",")
7          males.append(couples[0])
8          females.append(couples[1].rstrip())
9
```

Med keywordet **with** kalder vi det returnerede objekt fra **open** metoden **f**

Vi løber igennem hver linje i filen **f**

Vi siger at på hver linje i filen vi får, skal vi splitte linjen op ved kommaet

Couples er nu en liste, hvor vi ved at venstre side af kommaet er mand

- Live!

Det sidste element der er i listen, bruger vi metoden **rstrip** til at fjerne støj, fordi i filer bliver der tilføjet "**\n**" til slut i linjen for at skifte til næste linje

Opgaver pt. 2 (30 minutter)

1. Download filen:

https://raw.githubusercontent.com/AndersBensen/python_101/main/python2/points.txt

Og læs den ind i en *x* liste og en *y* liste med `open` metoden

- Hints:

- *x* → venstre side af kommaet, *y* → højre side af kommaet
- Drag inspiration fra *couples.txt* eksemplet
- Brug *float* metoden til at konvertere om til float når du tilføjer til listerne
 - Du behøver ikke bruge *rstrip* metoden når du konverterer til float! Den fjerner selv "\n"

2. Lav en liste med dine 5 yndlings øl (eller vine, eller sodavand, osv...) og skriv dem til filen *favourite_beverage.txt*

3. (Ekstra) Brug input metoden så en bruger i konsollen kan skrive sine 5 yndlingsøl, når de 5 øl er skrevet skal du så gemme dem i en fil *user_fav_bev.txt*

Biblioteker

Biblioteker

- Et bibliotek er en samling af kode som nogle andre har programmeret til et specifikt formål
 - Hvorfor opfinde den dybe tallerken hvis den allerede findes?
- Vi gider nemlig ikke selv at lave alting fra bunden, hvis der allerede er nogle dygtige mennesker der har dedikeret sig til det
- Så vi kan altså bruge andres bibliotek i vores kode, hvis vi skal bruge det til noget specifikt
- Biblioteker er et generelt koncept, og gælder i stort set alle programmeringssprog!

Biblioteker i Python

- Python er kendt for sine utrolig mange biblioteker (mere end 140.000+)
 - Det er noget af det der gør Python så fedt!
 - Et bibliotek kaldes også en ”package” (pakke)
- Der findes biblioteker til stort set alt f.eks.:
 - Machine learning (*sklearn*)
 - Lineær algebra (*numpy*)
 - HTTP (netværk) server (*flask*)
 - Visualisering og plotning af data (*matplotlib*)
 - Grafisk brugergrænseflade (*tkinter*)

pip

- Python har ret mange biblioteker liggende som standard f.eks. *math* og *tkinter*
- Men ikke alle biblioteker er der!
 - Derfor pip
 - Som er et program der er lavet til at installere biblioteker nemt
 - I fik det med da i downloadede Python (forhåbentlig)
- Så biblioteker der findes til python kan installeres vha. pip
 - I en kommandoprompt
 - Vi får se hvordan...

matplotlib

- Et bibliotek (en pakke) lavet til at ”plotte” data!
- Indeholder afsindigt mange metoder til at plotte data på alle mulige sjove og mærkelige måder
 - Histogram, 2d scatter, 3d scatter, logaritmisk osv ...
- Er en standard del af en hver Data Scientists værktøjskasse

matplotlib

```
1  import matplotlib.pyplot as plt
2
3  x = [1,2,3,4]
4  y = [2,4,8,16]
5
6  plt.plot(x,y)
7  plt.show()
```

matplotlib

```
1 import matplotlib.pyplot as plt
2
3 x = [1,2,3,4]
4 y = [2,4,8,16]
5
6 plt.plot(x,y)
7 plt.show()
```

Vi importerer filen **pyplot** fra pakken **matplotlib** og navngiver det **plt**, senere kan vi så kalde metoderne på **plt**

matplotlib

```
1 import matplotlib.pyplot as plt
2
3 x = [1,2,3,4]
4 y = [2,4,8,16]
5
6 plt.plot(x,y)
7 plt.show()
```

Vi importerer filen **pyplot** fra pakken **matplotlib** og navngiver det **plt**, senere kan vi så kalde metoderne på **plt**

Vi bruger **plot** metoden fra **plt** filen, og giver den vores x og y liste med

matplotlib

```
1 import matplotlib.pyplot as plt
2
3 x = [1,2,3,4]
4 y = [2,4,8,16]
5
6 plt.plot(x,y)
7 plt.show()
```

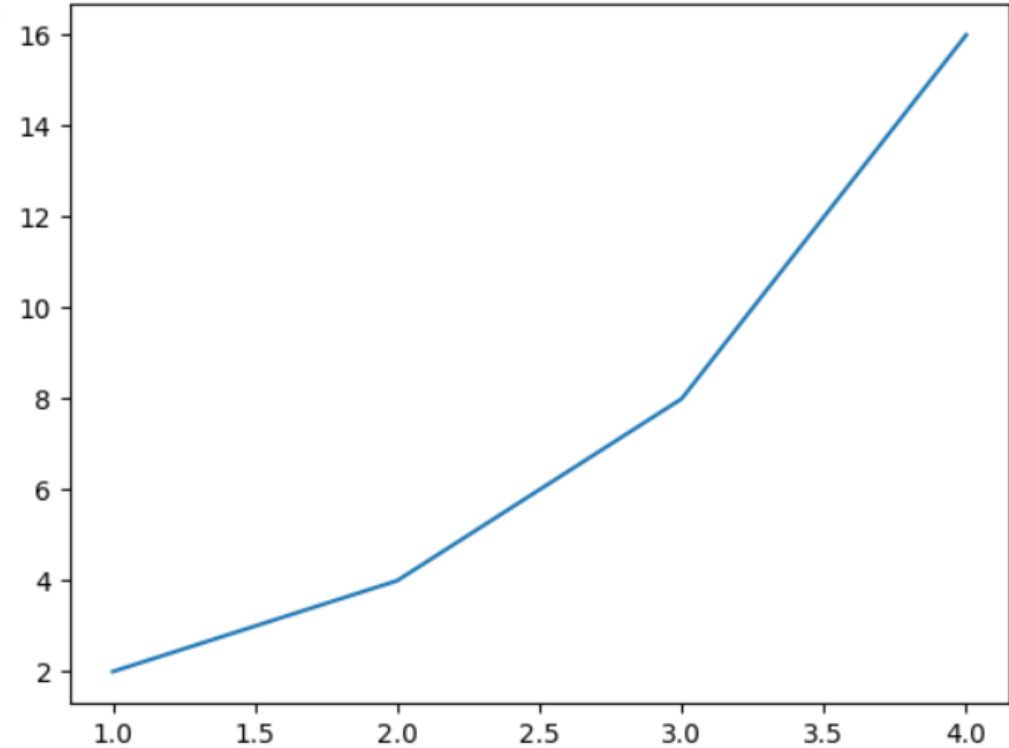
Vi importerer filen **pyplot** fra pakken **matplotlib** og navngiver det **plt**, senere kan vi så kalde metoderne på **plt**

Vi bruger **plot** metoden fra **plt** filen, og giver den vores x og y liste med

Vi bruger **show** metoden fra **plt** filen for at få vores plot frem!

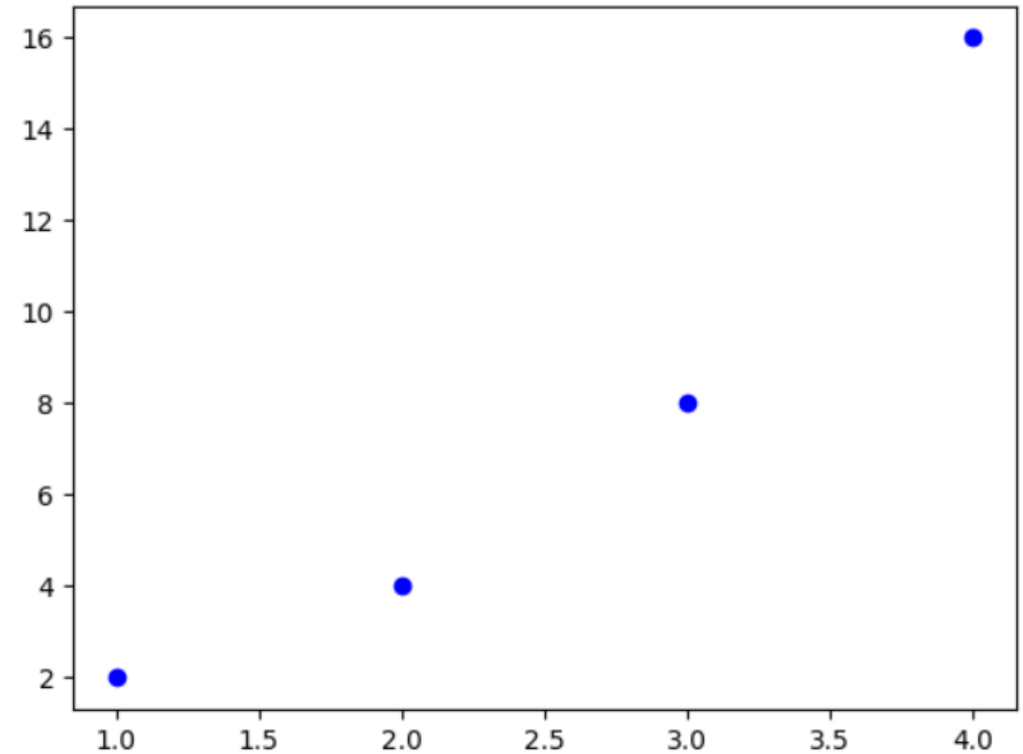
matplotlib

```
1  import matplotlib.pyplot as plt
2
3  x = [1,2,3,4]
4  y = [2,4,8,16]
5
6  plt.plot(x,y)
7  plt.show()
```



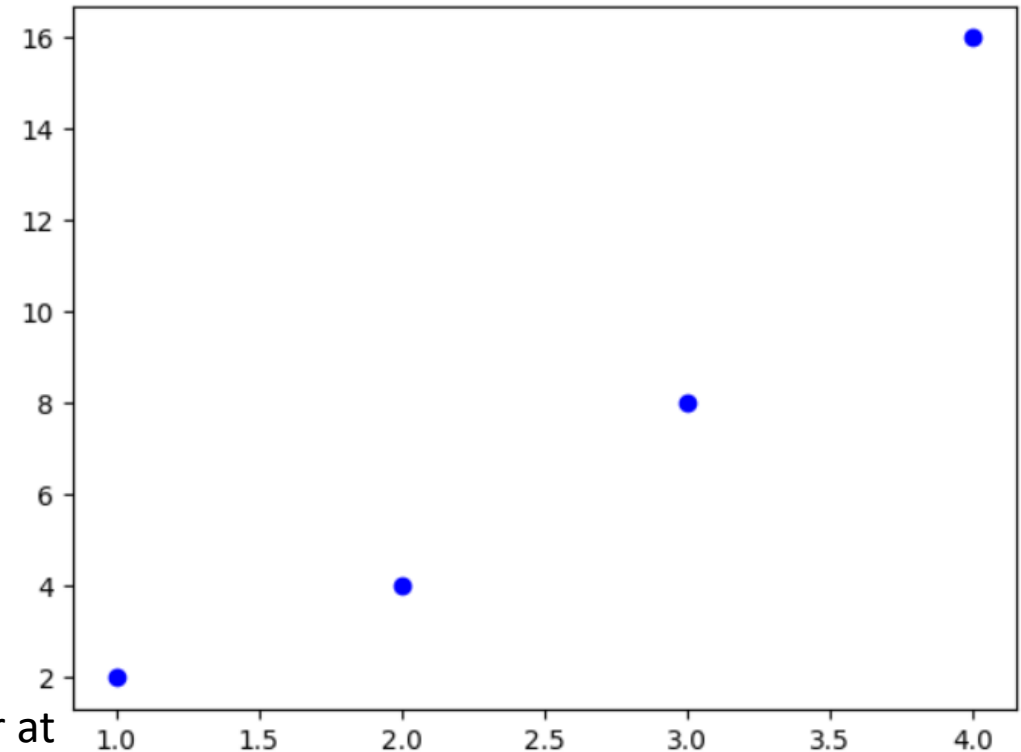
matplotlib

```
1  import matplotlib.pyplot as plt
2
3  x = [1,2,3,4]
4  y = [2,4,8,16]
5
6  plt.plot(x,y, 'bo')
7  plt.show()
```



matplotlib

```
1 import matplotlib.pyplot as plt
2
3 x = [1,2,3,4]
4 y = [2,4,8,16]
5
6 plt.plot(x,y, 'bo')
7 plt.show()
```

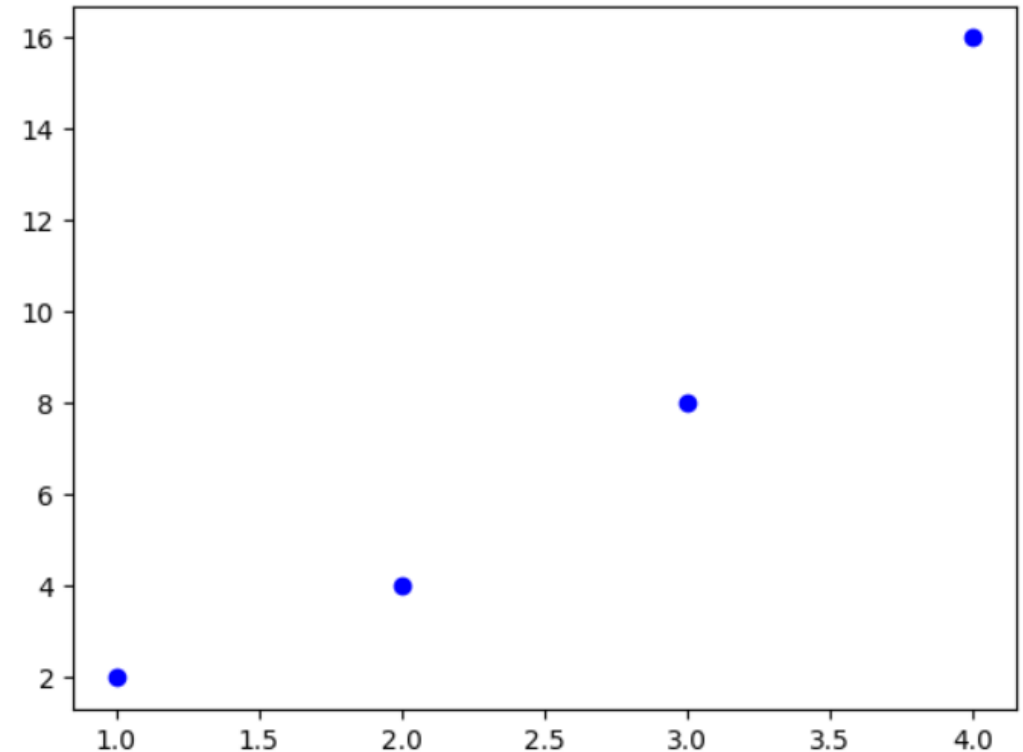


Så vi kan give et argument med, for at sige hvordan plottet skal se ud. **'bo'** betyder blå prikker.

matplotlib

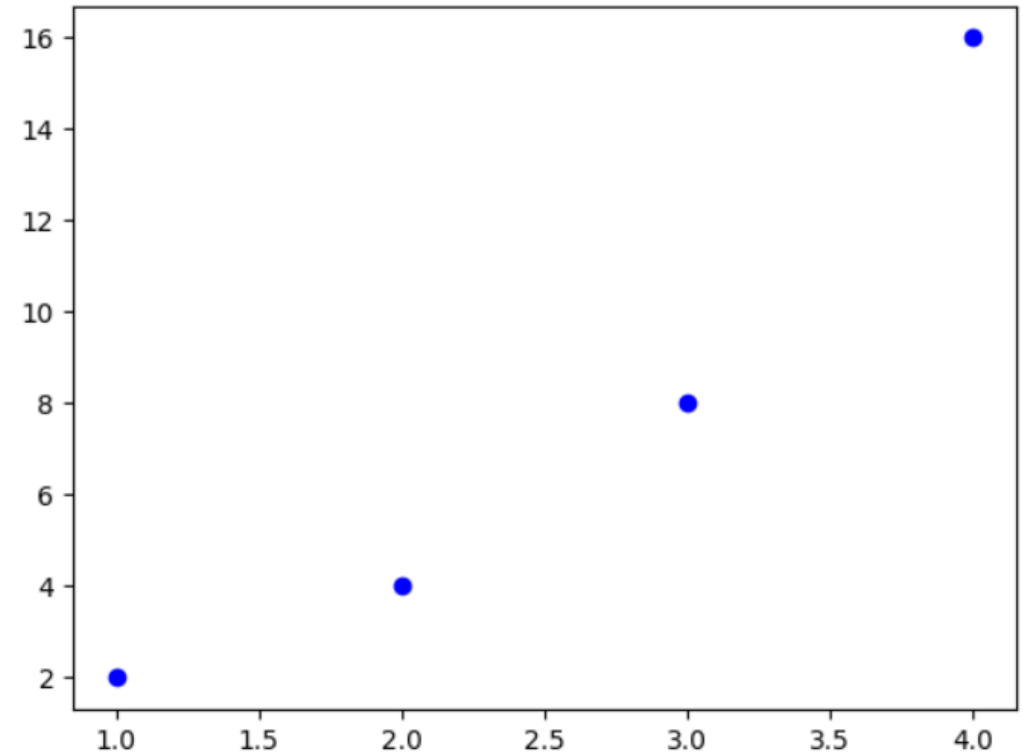
```
1  import matplotlib.pyplot as plt
2
3  x = [1,2,3,4]
4  y = [2,4,8,16]
5
6  plt.plot(x,y, 'bo')
7  plt.show()
```

- Men hvordan downloader vi matplotlib?
 - *pip install matplotlib*
 - Nogle gange (mac?): *pip3 install matplotlib*



matplotlib

```
1  import matplotlib.pyplot as plt
2
3  x = [1,2,3,4]
4  y = [2,4,8,16]
5
6  plt.plot(x,y, 'bo')
7  plt.show()
```



- Men hvordan downloader vi matplotlib?
 - *pip install matplotlib*
 - Nogle gange (mac?): *pip3 install matplotlib*
- Live!

Opgaver pt. 3 (20 minutter)

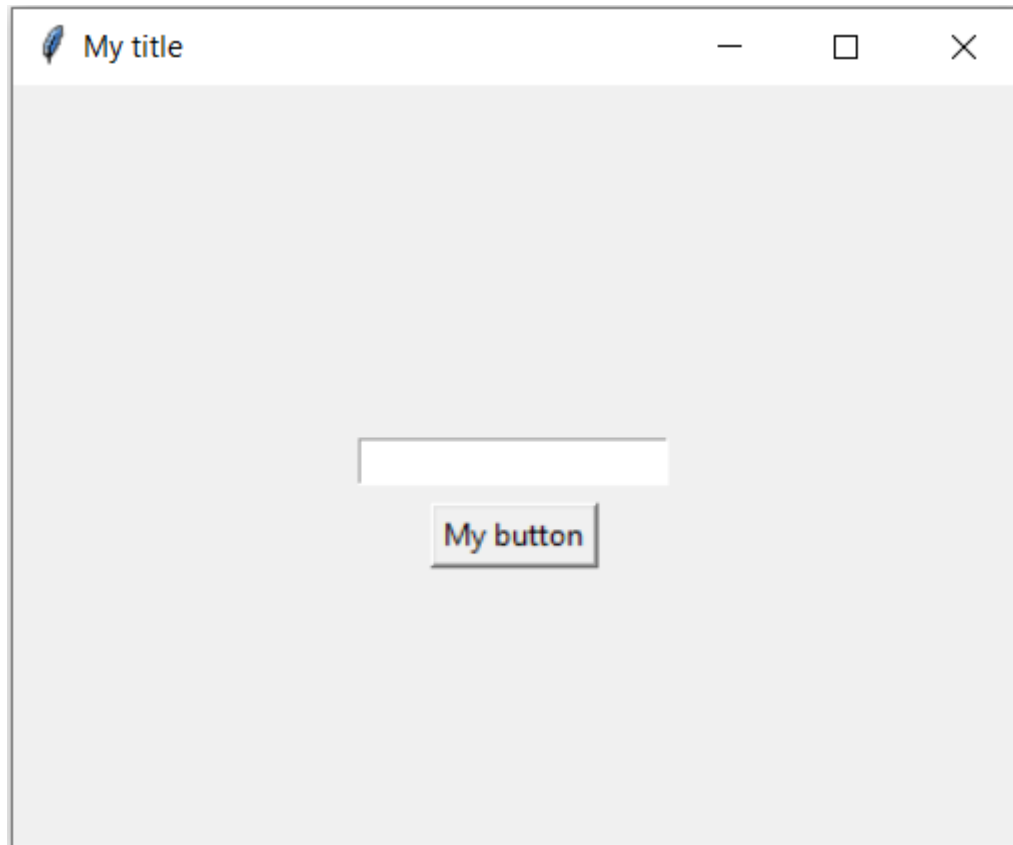
1. I opgave 2.1 læste i min *points.txt* fil ind i to lister: *x* og *y*. Brug nu biblioteket *matplotlib.pyplot* til at plotte de to lister! Prikkerne skal være røde.
 - Hints:
 - Skriv kommandoen: *"pip install matplotlib"* i jeres kommandoprompt for at installere *matplotlib*
 - På mac: *"pip3 install matplotlib"*
 - Hint: *"ro"* giver røde prikker
2. Brug *matplotlib*'s indbyggede metoder til at give plot titlen 'One of Anscombes quartet', x-akse titlen 'x' og y-akse titlen 'y'
 - Hints:
 - Metoderne hedder 'title', 'xlabel', 'ylabel'
3. (Ekstra) Hvis i ikke skriver *"ro"* som argument til metoden, men kun giver de to lister, hvorfor ser plottet så, så mærkeligt ud?
 - Hints:
 - Prøv at kigge i *points.txt* filen

Grafisk brugergrænseflade

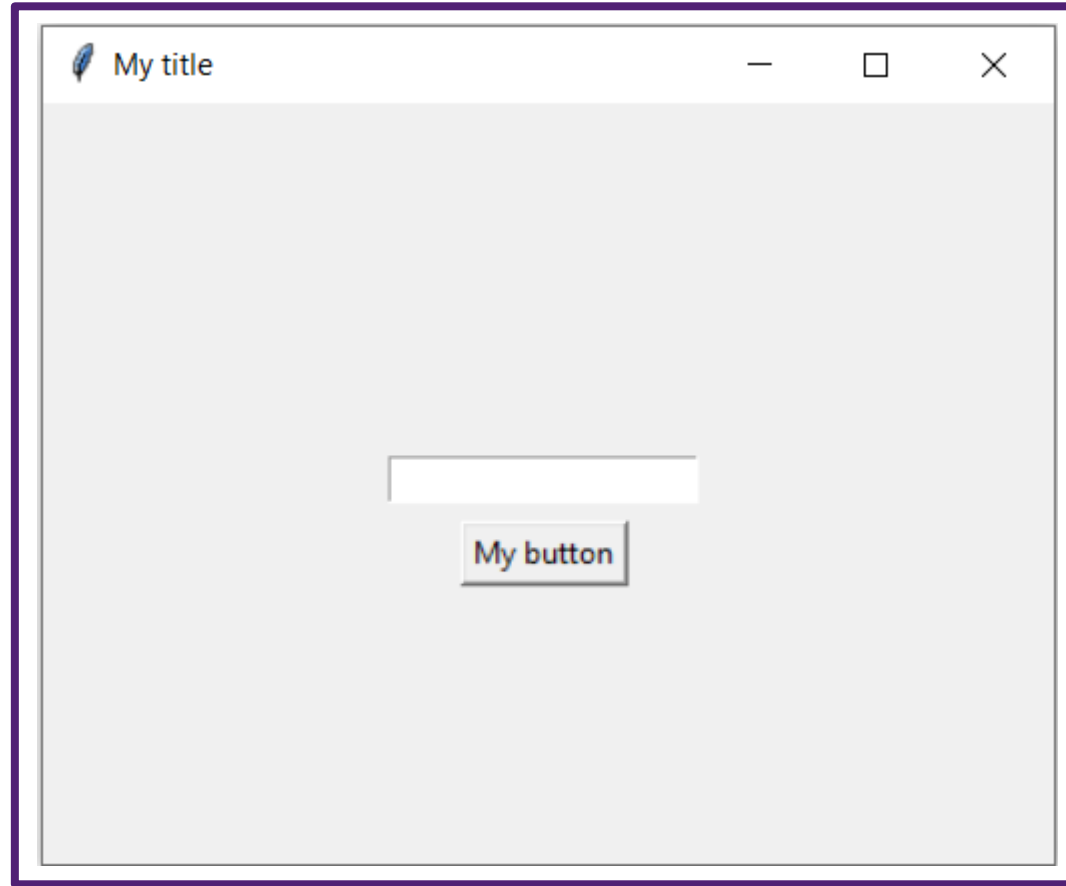
Grafisk brugergrænseflade

- Kaldes også for GUI (Graphical User Interface)
- Indtil videre har vi set hvordan man skriver programmer som kører i en kommandoprompt
 - Og måske kan man tale med dem igennem input metoden
- Alle de programmer i kender og elsker har jo noget grafisk foran, som man interagerer med
 - Det kan vi selvfølgelig også lave i python!

GUI terminology

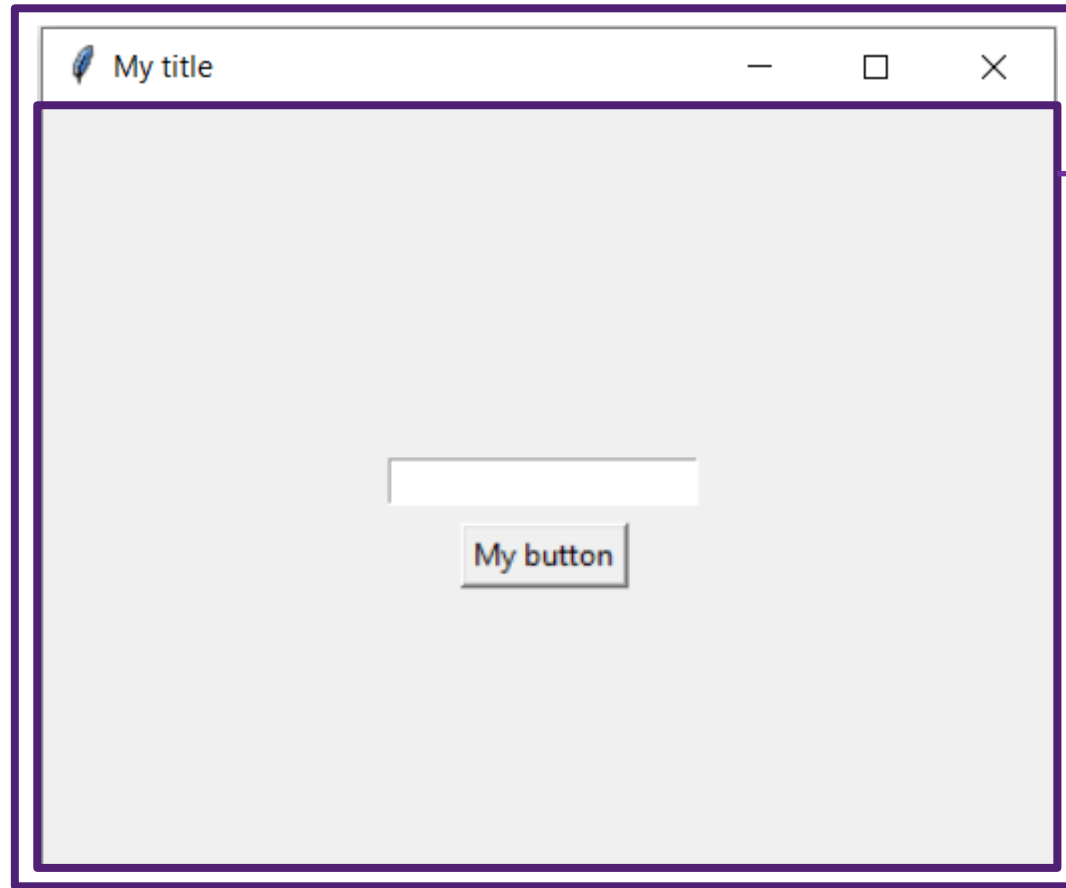


GUI terminologi



Window → Selve
vinduet hvor alting er
placeret inde i

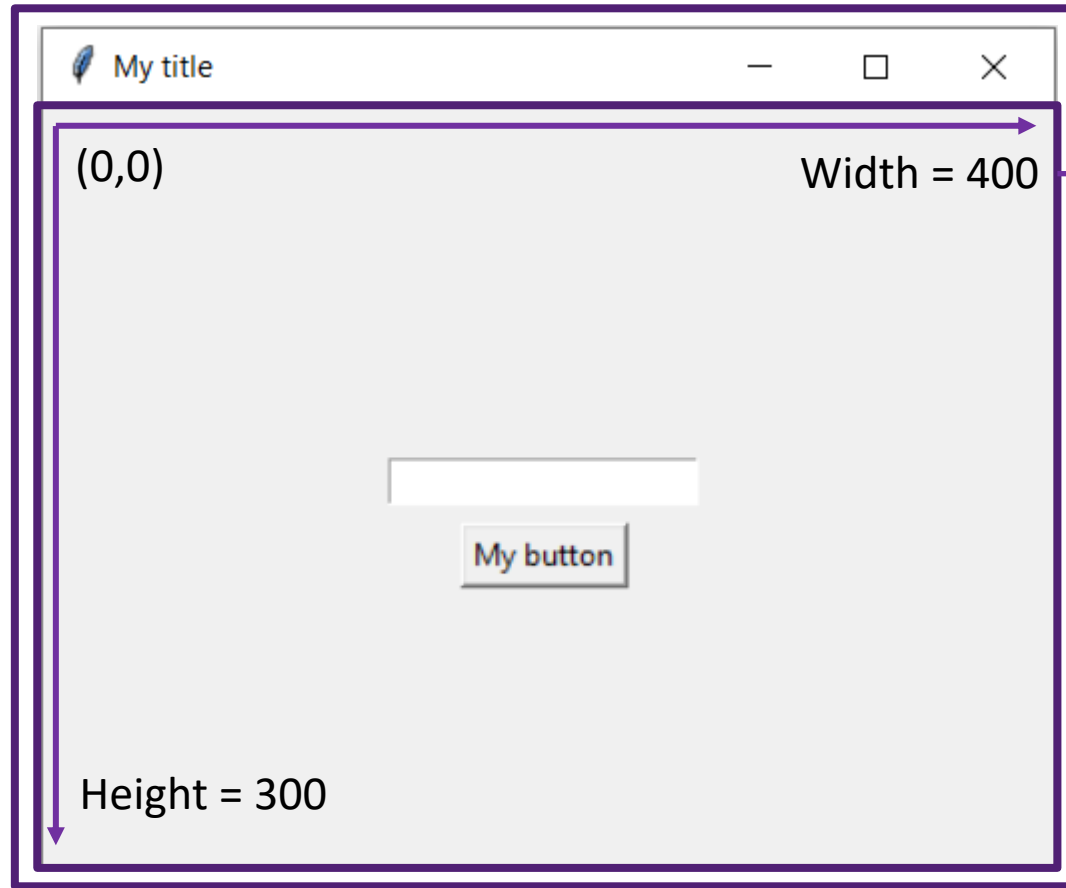
GUI terminologi



Canvas → Der hvor vi placerer indholdet i vinduet

Window → Selve vinduet hvor alting er placeret inde i

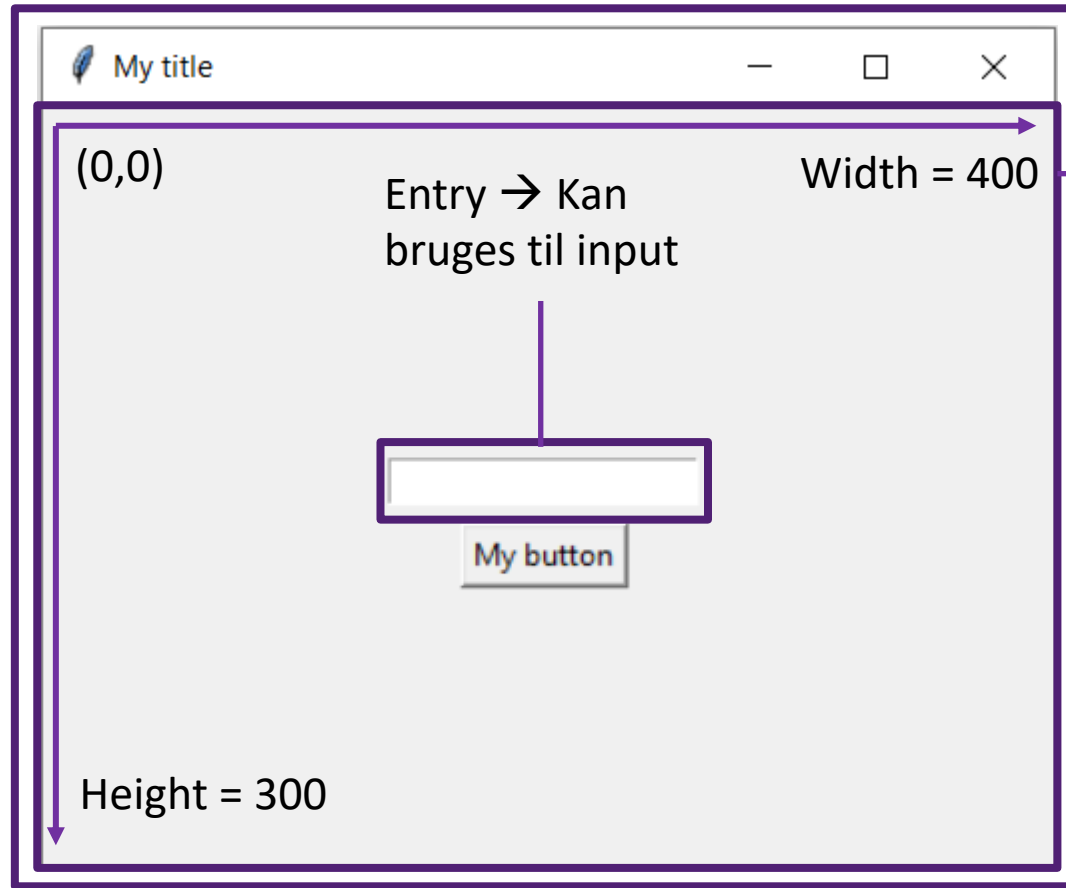
GUI terminologi



Canvas → Der hvor vi placerer indholdet i vinduet

Window → Selve vinduet hvor alting er placeret inde i

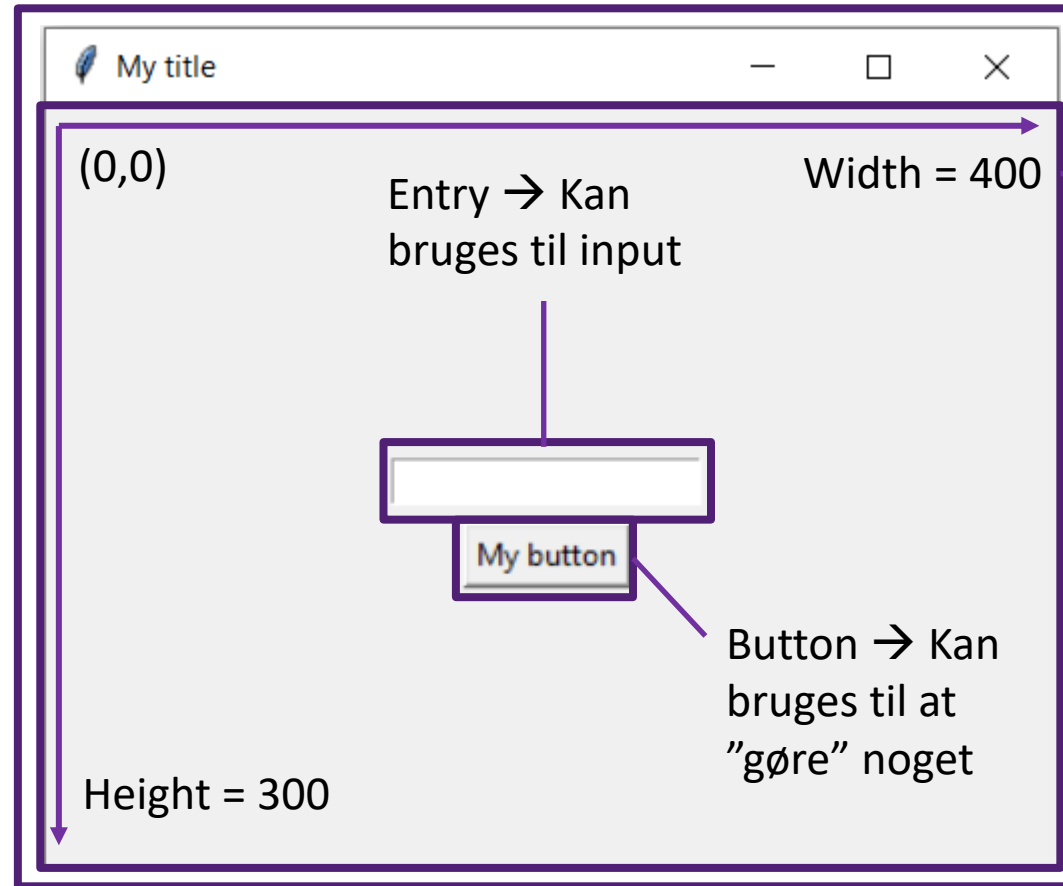
GUI terminologi



Canvas → Der hvor vi placerer indholdet i vinduet

Window → Selve vinduet hvor alting er placeret inde i

GUI terminologi



Entry → Kan
bruges til input

Width = 400

(0,0)

Height = 300

My button

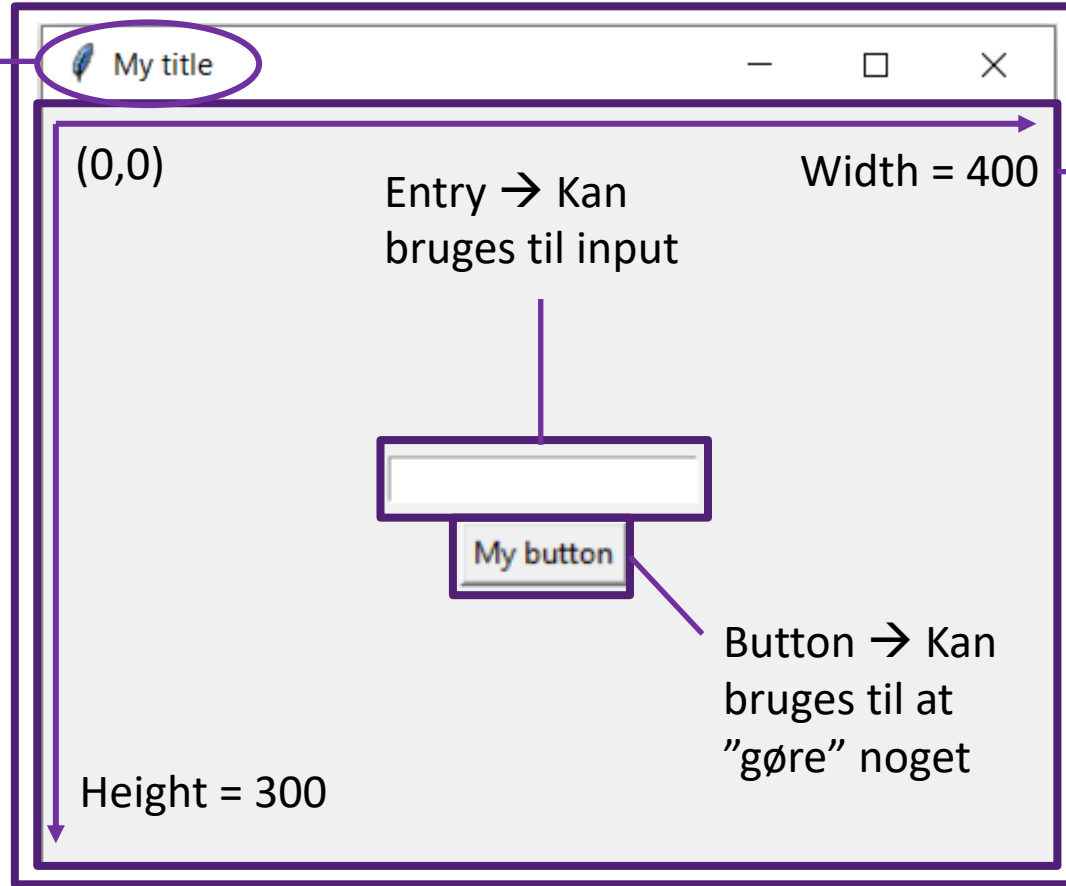
Button → Kan
bruges til at
"gøre" noget

Canvas → Der hvor vi
placrer indholdet i
vinduet

Window → Selve
vinduet hvor alting er
placeret inde i

GUI terminologi

Title → Selve
titlen på vores
vindue



Canvas → Der hvor vi
placerer indholdet i
vinduet

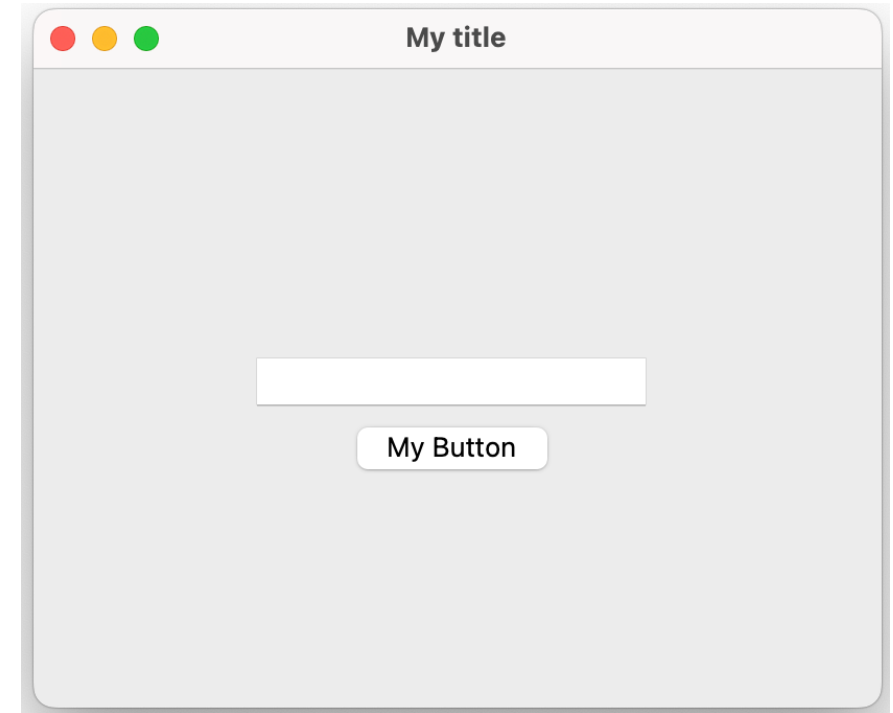
Window → Selve
vinduet hvor alting er
placeret inde i

Tkinter

- Et bibliotek der er lavet til at lave GUI i Python
- Er bygget objekt-orienteret
 - Hvilket I nu ved hvad betyder!
- Ligger allerede som standard inde i Python
 - Så vi behøver ikke engang pip

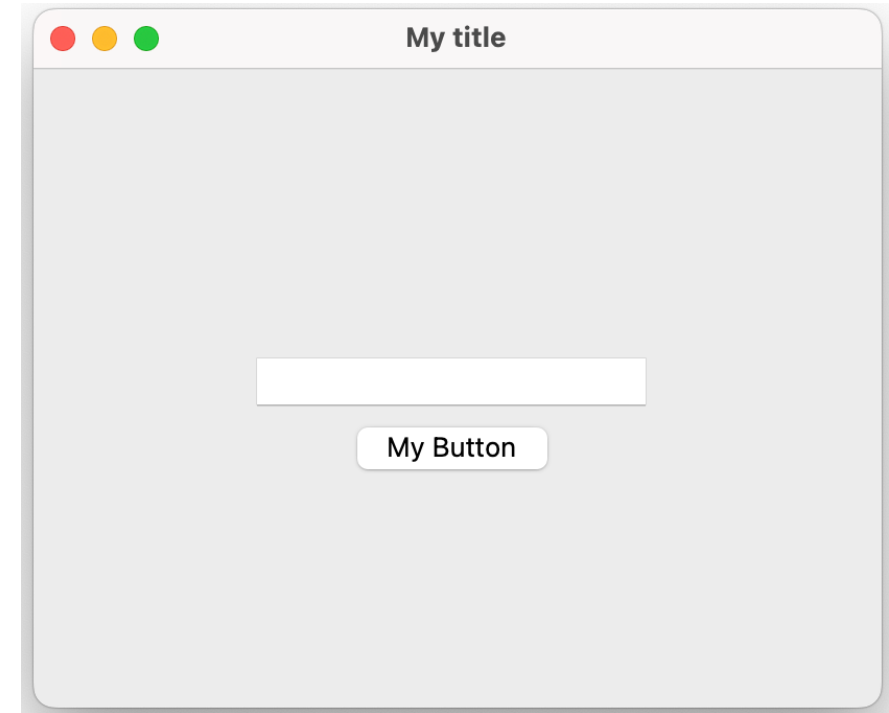
Tkinter

```
1  import tkinter as tk
2
3  window = tk.Tk()
4  window.title("My title")
5
6  canvas = tk.Canvas(window, width = 400, height = 300)
7  canvas.pack()
8
9  entry = tk.Entry(window)
10 canvas.create_window(200, 150, window=entry)
11
12 button = tk.Button(text='My button')
13 canvas.create_window(200, 180, window=button)
14
15 window.mainloop()
```



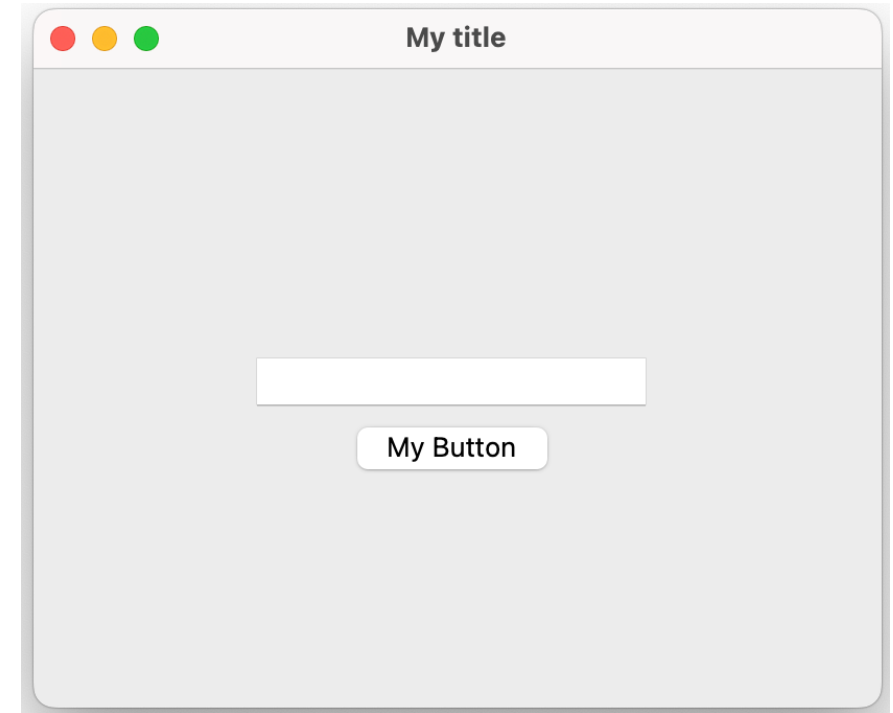
Tkinter

```
1 import tkinter as tk
2
3 window = tk.Tk()
4 window.title("My title")
5
6 canvas = tk.Canvas(window, width = 400, height = 300)
7 canvas.pack()
8
9 entry = tk.Entry(window)
10 canvas.create_window(200, 150, window=entry)
11
12 button = tk.Button(text='My button')
13 canvas.create_window(200, 180, window=button)
14
15 window.mainloop()
```



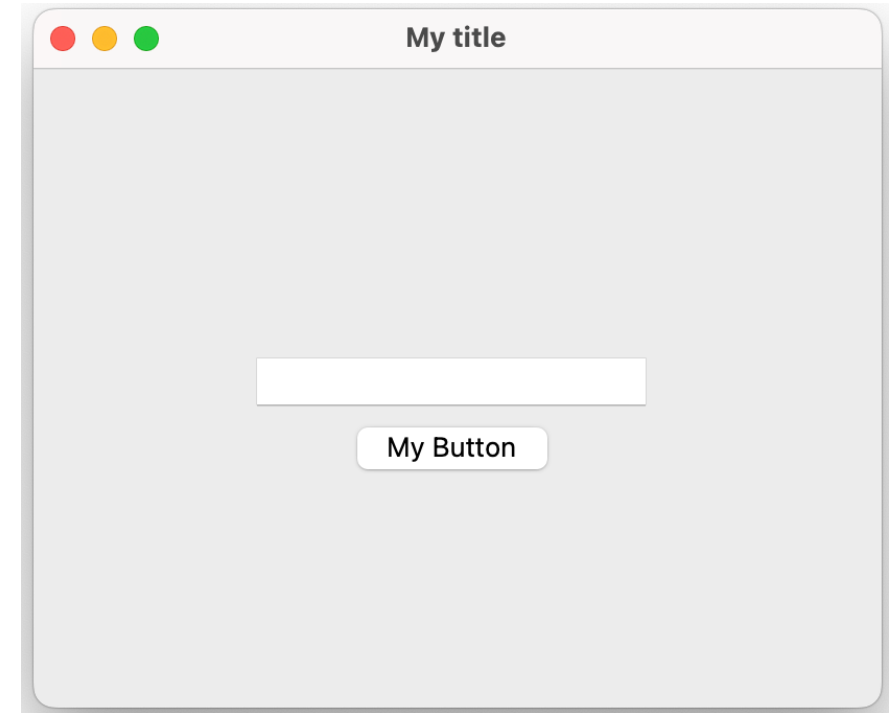
Tkinter

```
1 import tkinter as tk
2
3 window = tk.Tk()
4 window.title("My title")
5
6 canvas = tk.Canvas(window, width = 400, height = 300)
7 canvas.pack()
8
9 entry = tk.Entry(window)
10 canvas.create_window(200, 150, window=entry)
11
12 button = tk.Button(text='My button')
13 canvas.create_window(200, 180, window=button)
14
15 window.mainloop()
```



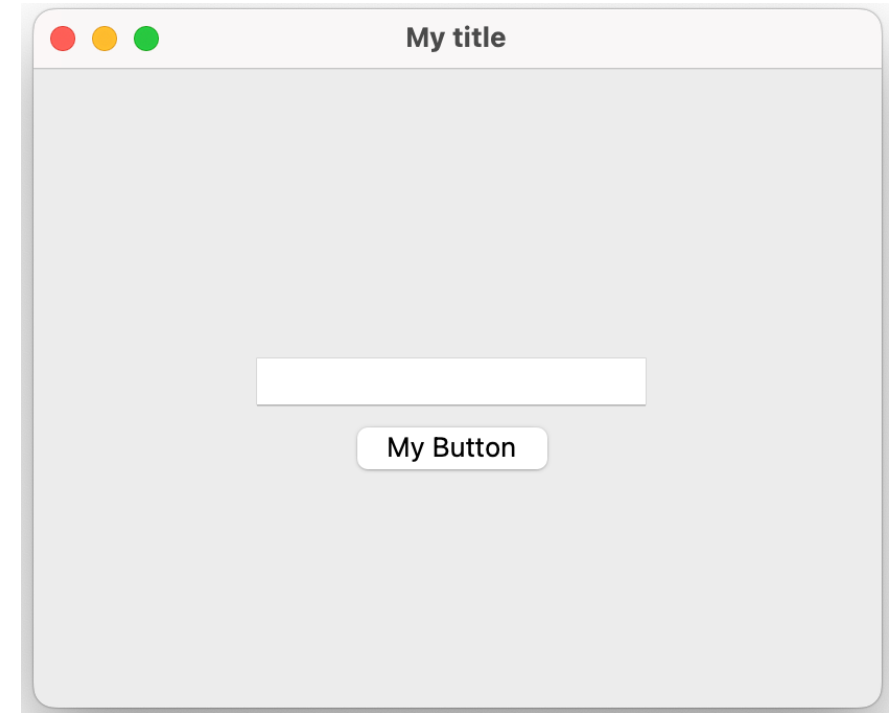
Tkinter

```
1 import tkinter as tk
2
3 window = tk.Tk()
4 window.title("My title")
5
6 canvas = tk.Canvas(window, width = 400, height = 300)
7 canvas.pack()
8
9 entry = tk.Entry(window)
10 canvas.create_window(200, 150, window=entry)
11
12 button = tk.Button(text='My button')
13 canvas.create_window(200, 180, window=button)
14
15 window.mainloop()
```



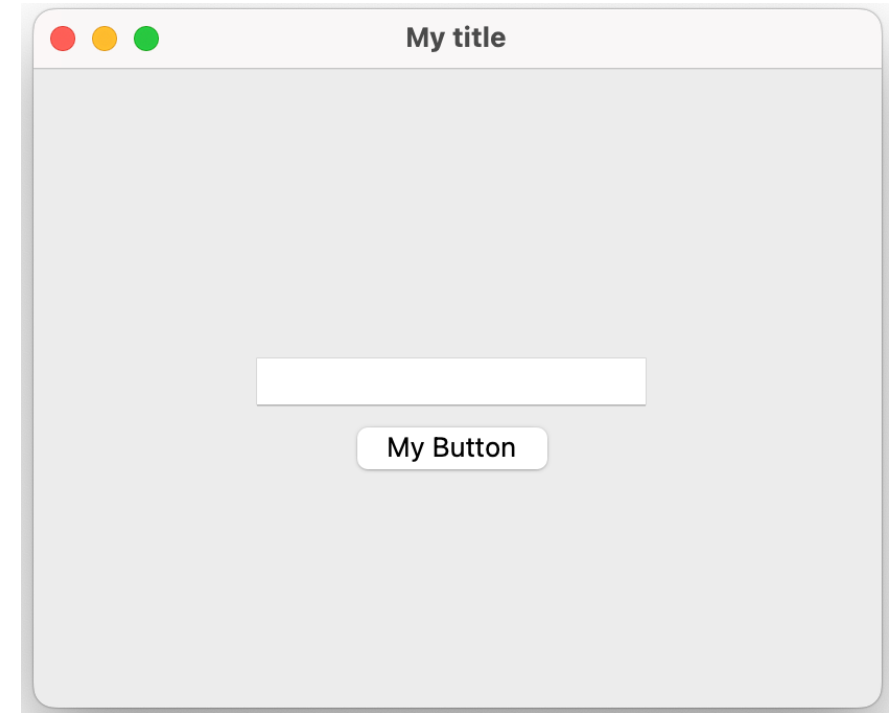
Tkinter

```
1 import tkinter as tk
2
3 window = tk.Tk()
4 window.title("My title")
5
6 canvas = tk.Canvas(window, width = 400, height = 300)
7 canvas.pack()
8
9 entry = tk.Entry(window)
10 canvas.create_window(200, 150, window=entry)
11
12 button = tk.Button(text='My button')
13 canvas.create_window(200, 180, window=button)
14
15 window.mainloop()
```



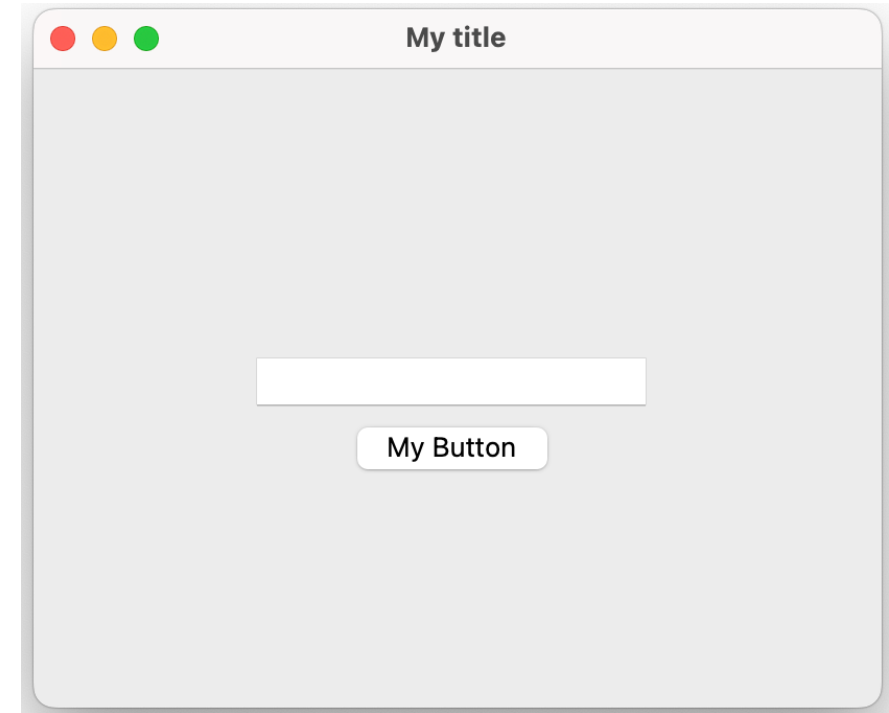
Tkinter

```
1 import tkinter as tk
2
3 window = tk.Tk()
4 window.title("My title")
5
6 canvas = tk.Canvas(window, width = 400, height = 300)
7 canvas.pack()
8
9 entry = tk.Entry(window)
10 canvas.create_window(200, 150, window=entry)
11
12 button = tk.Button(text='My button')
13 canvas.create_window(200, 180, window=button)
14
15 window.mainloop()
```



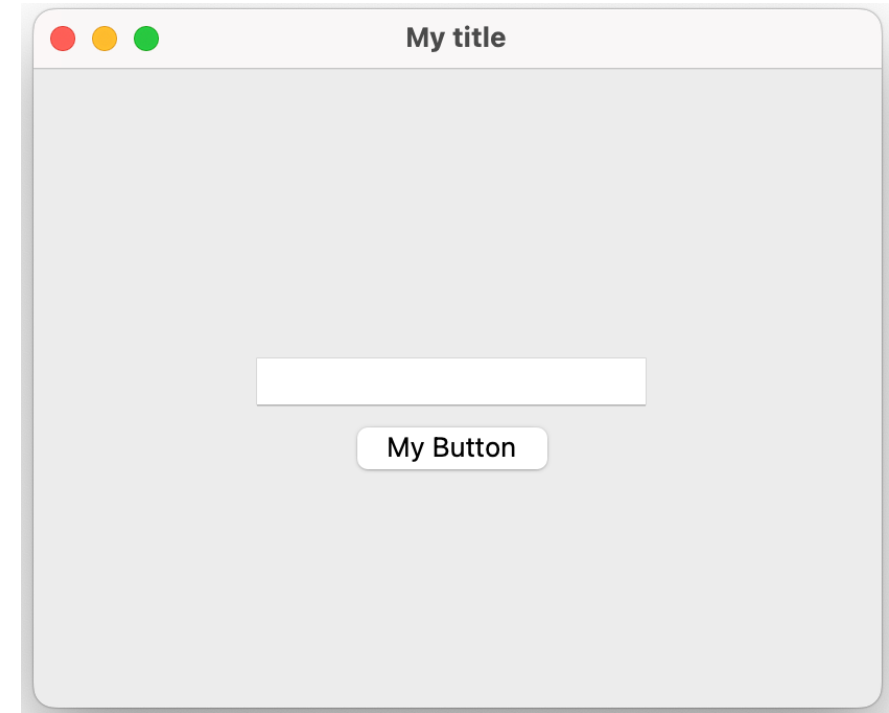
Tkinter

```
1 import tkinter as tk
2
3 window = tk.Tk()
4 window.title("My title")
5
6 canvas = tk.Canvas(window, width = 400, height = 300)
7 canvas.pack()
8
9 entry = tk.Entry(window)
10 canvas.create_window(200, 150, window=entry)
11
12 button = tk.Button(text='My button')
13 canvas.create_window(200, 180, window=button)
14
15 window.mainloop()
```



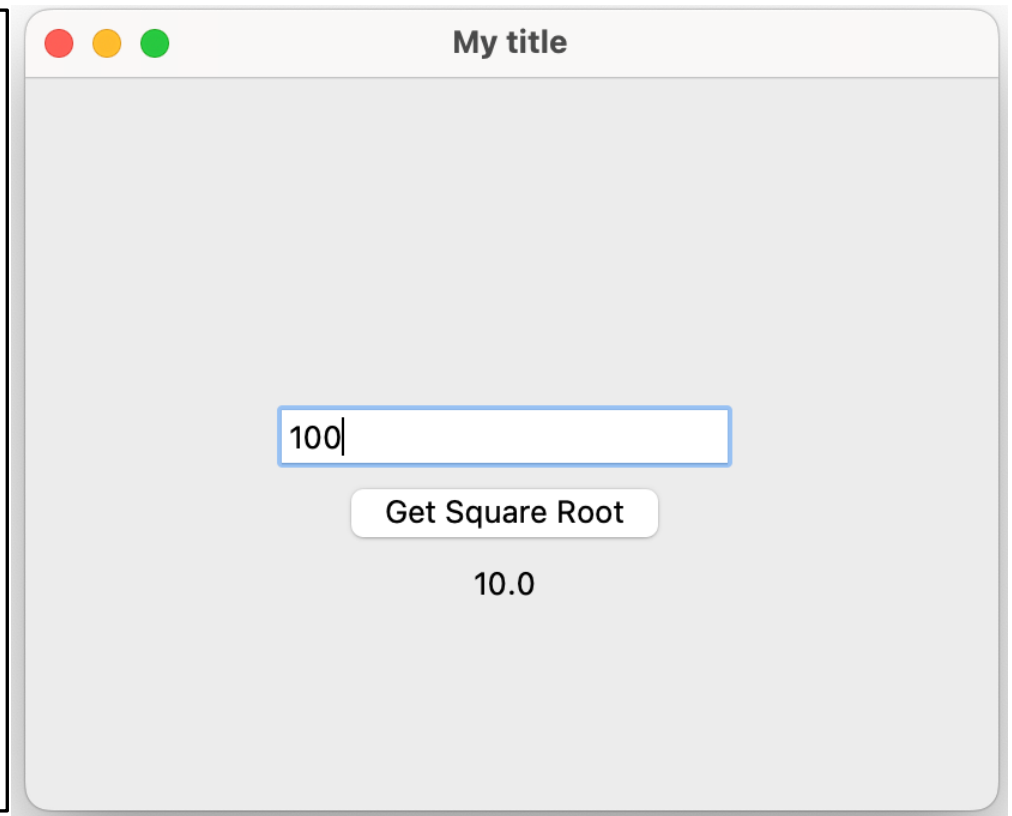
Tkinter

```
1 import tkinter as tk
2
3 window = tk.Tk()
4 window.title("My title")
5
6 canvas = tk.Canvas(window, width = 400, height = 300)
7 canvas.pack()
8
9 entry = tk.Entry(window)
10 canvas.create_window(200, 150, window=entry)
11
12 button = tk.Button(text='My button')
13 canvas.create_window(200, 180, window=button)
14
15 window.mainloop()
```



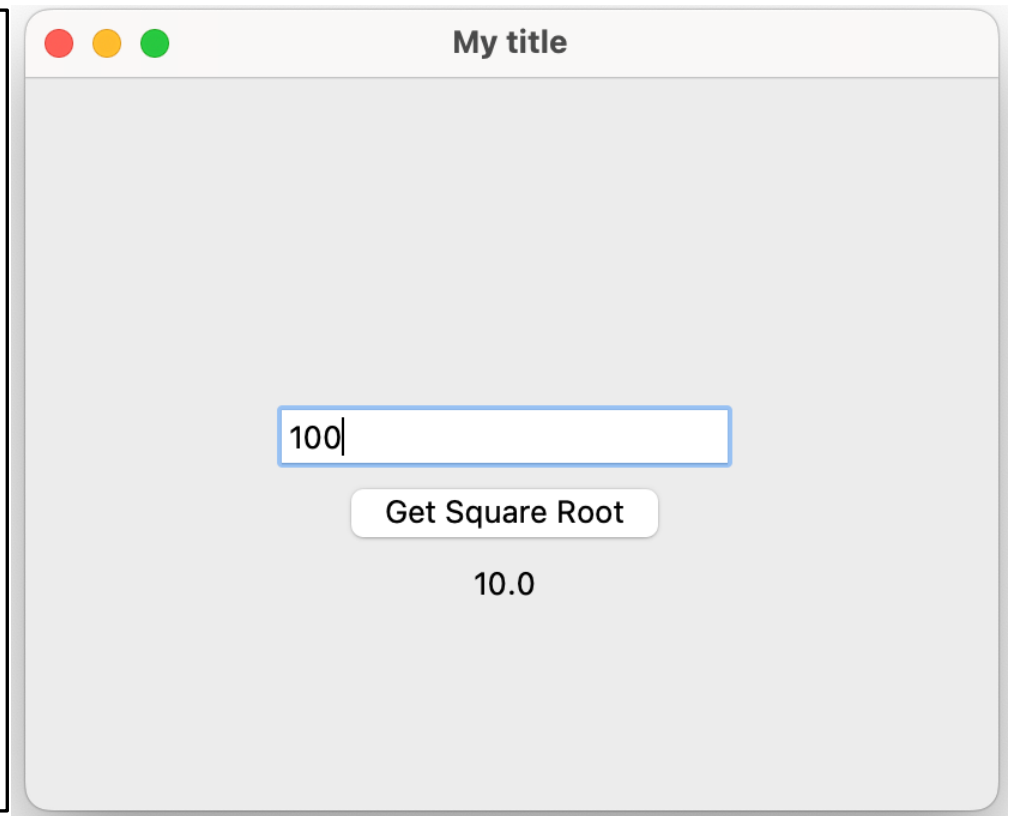
Tkinter

```
1  import tkinter as tk
2  import math
3
4  window = tk.Tk()
5  window.title("My title")
6
7  canvas = tk.Canvas(window, width=400, height=300)
8  canvas.pack()
9
10 entry = tk.Entry(window)
11 canvas.create_window(200, 150, window=entry)
12
13 def get_square_root():
14     input = entry.get()
15     label = tk.Label(window, text=math.sqrt(int(input)))
16     canvas.create_window(200, 210, window=label)
17
18 button = tk.Button(text="Get Square Root", command=get_square_root)
19 canvas.create_window(200, 180, window=button)
20 window.mainloop()
```



Tkinter

```
1  import tkinter as tk
2  import math
3
4  window = tk.Tk()
5  window.title("My title")
6
7  canvas = tk.Canvas(window, width=400, height=300)
8  canvas.pack()
9
10 entry = tk.Entry(window)
11 canvas.create_window(200, 150, window=entry)
12
13 def get_square_root():
14     input = entry.get()
15     label = tk.Label(window, text=math.sqrt(int(input)))
16     canvas.create_window(200, 210, window=label)
17
18 button = tk.Button(text="Get Square Root", command=get_square_root)
19 canvas.create_window(200, 180, window=button)
20 window.mainloop()
```



- Live!

Opgaver pt. 4 (20 minutter)

1. Lav en GUI hvor brugeren kan skrive grader i celsius, og så får dem tilbage i fahrenheit

- Formlen er: *celsius_to_fahrenheit* = $(1.8 * celsius_degrees) + 32$
- Hints:
 - Lav en metode som i mit potens eksempel, men i stedet omregn til fahrenheit i stedet for potens
 - Husk at konverter dine celsius til *float* når du skal regne til fahrenheit

2. (Ekstra) Lav en GUI hvor brugeren kan skrive et "password", og når brugeren trykker på en knap så bliver det "password" gemt i en fil *passwords.txt*

- Hints:
 - Den metode du giver med som *command* til knappen, skal åbne en fil og skal "appende" passwordet til filen
 - Kig på de eksempler jeg gav med at skrive til en fil, men i stedet for "*w*" skal du skrive "*a*" som argument til *open* metoden

Avanceret emner

Machine learning

- Machine learning er sådan set bare statistiske/matematiske modeller der er programmeret, og man så bruger til at *forudse* ting
- Der er mange forskellige modeller
 - Decision tree
 - Logistisk regression
 - Naive bayes
 - K nearest neighbour (KNN)
 - Neurale netværk
 - Osv.

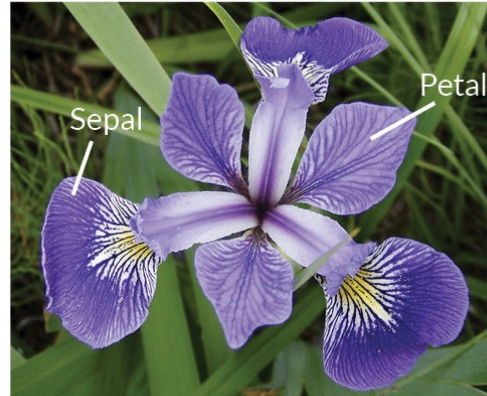
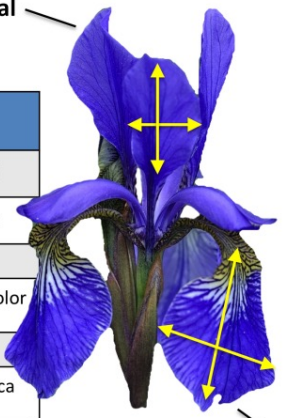
Machine learning case

Samples
(instances, observations)

	Sepal length	Sepal width	Petal length	Petal width	Class label
1	5.1	3.5	1.4	0.2	Setosa
2	4.9	3.0	1.4	0.2	Setosa
...					
50	6.4	3.5	4.5	1.2	Versicolor
...					
150	5.9	3.0	5.0	1.8	Virginica

Features
(attributes, measurements, dimensions)

Class labels
(targets)



Iris Versicolor



Iris Setosa



Iris Virginica

Afrunding

- Programmering er et håndværk
 - Man lærer det ved at gøre det
- Mere python:
 - <https://www.learnpython.org/>
- Og hvis I synes machine learning var spændende:
 - <https://scikit-learn.org/stable/tutorial/basic/tutorial.html>
 - https://pandas.pydata.org/pandas-docs/stable/user_guide/10min.html
 - Og artikler hvor det bliver brugt:
 - <https://medium.com/>
 - <https://towardsdatascience.com/>
- Alle løsninger (også kode til machine learning) til i dag kan findes her:
 - https://github.com/AndersBensen/python_101/raw/main/python2/exercises.zip

Spørgsmål?

- anders_bensen@hotmail.com