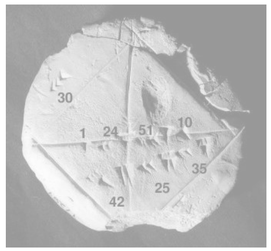
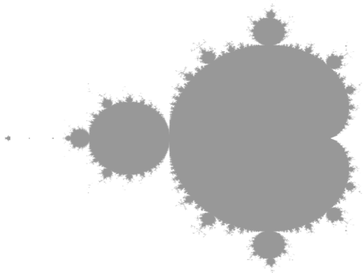
Beräkningsprogrammering i NumPy

* Delkurs på 1,5 hp till pythonkursen i programmering.
* Mycket av materialet och labbarna kommer från en Matlab-kurs gjord av Patrik Persson.
* Översatt till python: Maj Stenmark
* Vi kommer använda Pythonbibliotek (`numpy, scipy`) för att utföra beräkningar, t.ex.:
  * plotta grafer
  * lösa ekvationssystem och differentialekvationer
  * beräkna integraler
  * anpassa kurvor
* Python är ett populärt programmeringsspråk för vetenskapliga beräkningar (används för maskininlärning, datorseende, robotik …)

NumPy i EDAA85: upplägg våren 2024

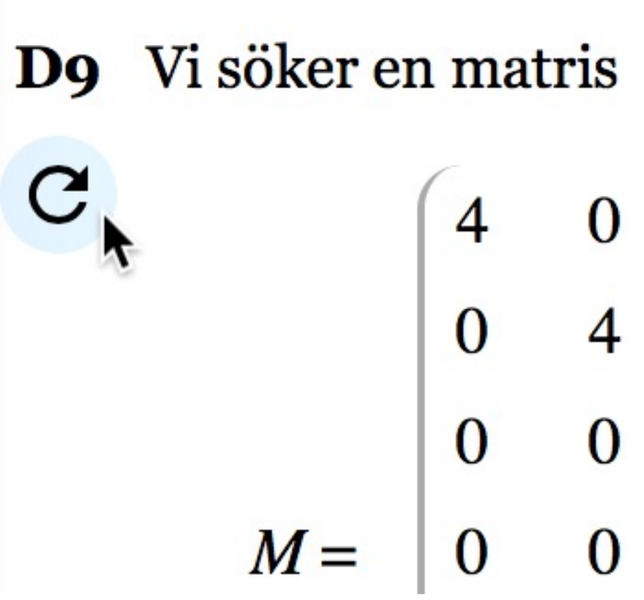
5-9 feb	26 feb - 3 mar	26 - 29 mar
Föreläsning	Föreläsning	Föreläsning
eget arbete	**eget arbete**	**eget arbete**
Labb 1	Labb 2	Labb 3





Laborationer: obligatoriska (3st)

* Webbaserade laborationer (länkar finns i Moodle)
* Vissa uppgifter genereras; ibland finns ledtrådar och/eller facit
* Ni genomför laborationen i förväg (i par om två studenter)
* Speciella schematillfällen.
* **För att bli godkänd:**
  övertyga handledaren om att du kan
  använda NymPy för att lösa uppgifterna.
* **Gör snygga utskrifter med resultat så att det är lätt att följa logiken i ditt program!**



Ditt arbete i NumPy-momentet

* 1,5hp = 40 timmar **självständigt arbete**
* **Förbered och genomför veckans laboration**
  - gärna i grupper om två studenter!

* **Redovisa på labbtiderna**

* **Material och hjälp:**
  - föreläsningar
  - läsanvisningar i boken *Scientific Computing with Python 3* av Claus Führer
  - resurstider
  - i chatten

Installation

* På universitetsdatorerna finns:
  - Python och nödvändiga bibliotek
  - Editorn VS Code med en Python-plugin.

* På Moodle finns installationsinstruktioner för Windows, Linux och Mac.

* Ni får gärna använda pakethanderaren Anaconda, *virtual environment* eller en annan editor.

Kommentar Lab 1

* Ni kan ha all kod i en pythonfil och sen skriva ut varje uppgift för sig: t ex

```
############################## Uppgift 2 ##################################
pi =  3.141592653589793
Roten ur 3 =  1.732050807568877 2
Roten ur -3 =  1.732050807568877 2j
Roten ur i =  (0.7071067811865476+0.7071067811865475j)
e ^ (pi * i) =  (-1+1.2246467991473532e-16j)
###########################################################################
```

Ditt arbete i NumPy-momentet

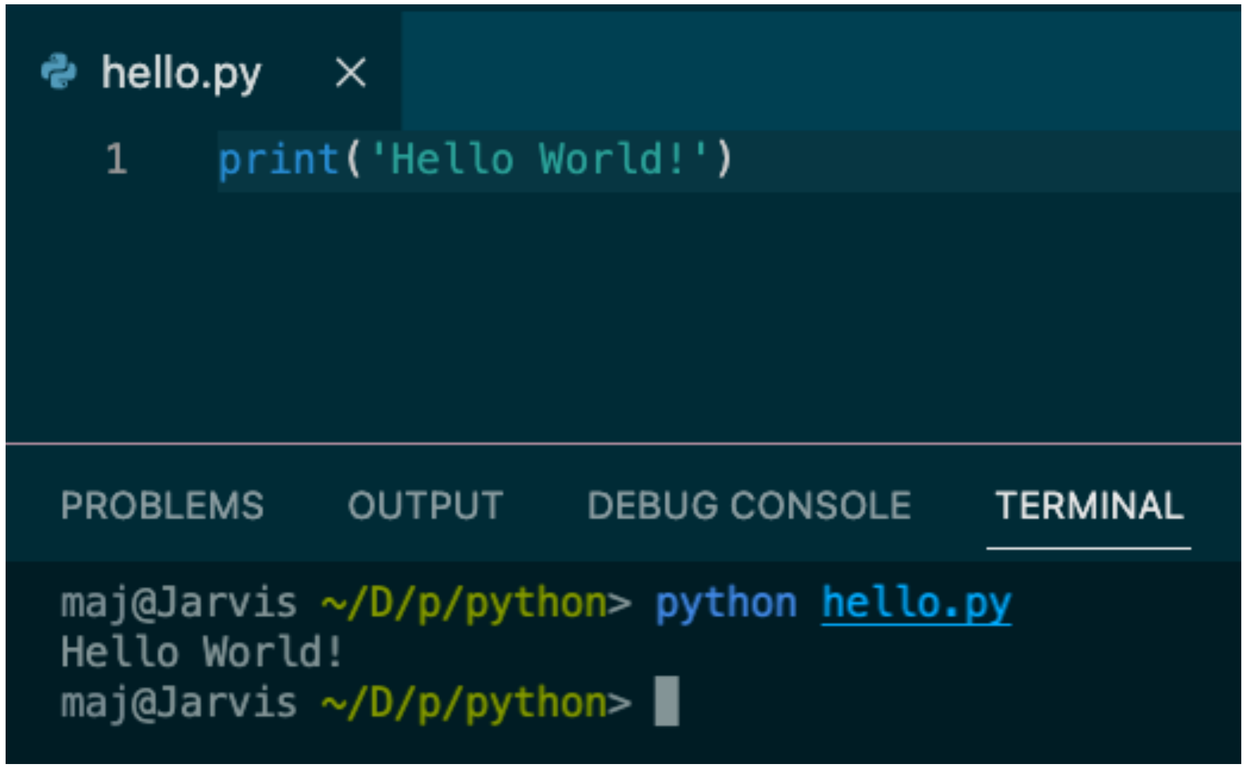
* 1,5hp = 40 timmar **självständigt arbete**
* **Förbered och genomför veckans laboration**
  - gärna i grupper om två studenter!

* **Redovisa på labbtiderna**

* **Material och hjälp:**
  - föreläsningar
  - läsanvisningar i boken *Scientific Computing with Python 3* av Claus Führer
  - resurstider
  - i Mattermost

Vecka 1

* python-repl:en och scriptfiler
* Grundläggande beräkningar
* Några standardfunktioner
* Definiera egna funktioner
* Vektorer och matriser
* Ekvationssystem
* Plotta grafer



Installation

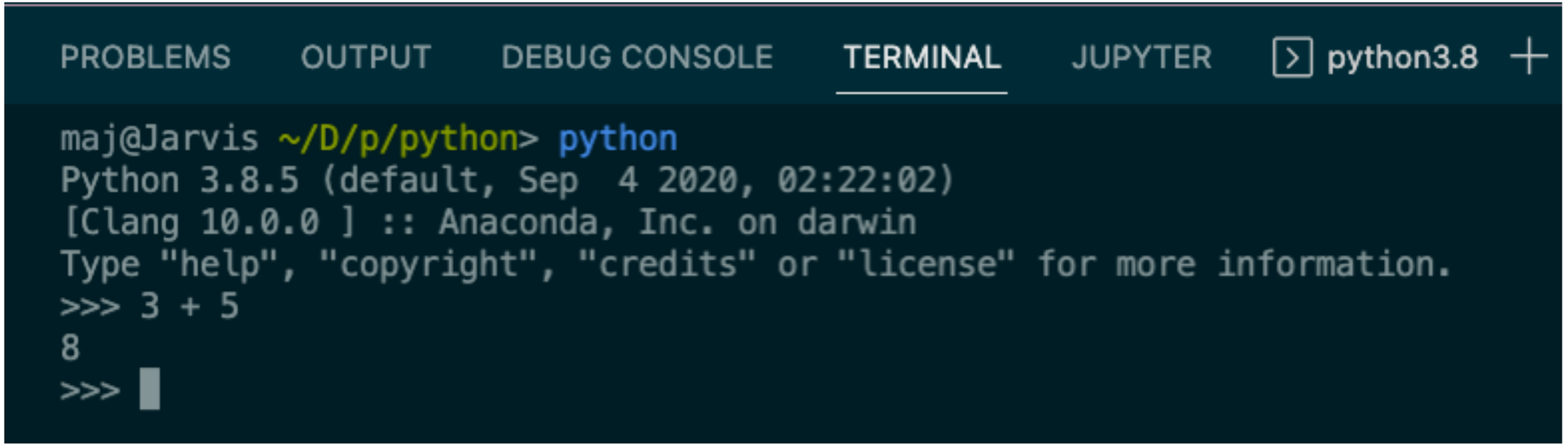
* På universitetsdatorerna finns:
  - Python och nödvändiga bibliotek
  - Editorn VS Code med en Python-plugin.

* På Moodle finns installationsinstruktioner för Windows, Linux och Mac.

* Ni får gärna använda pakethanderaren Anaconda, *virtual environment* eller en annan editor.

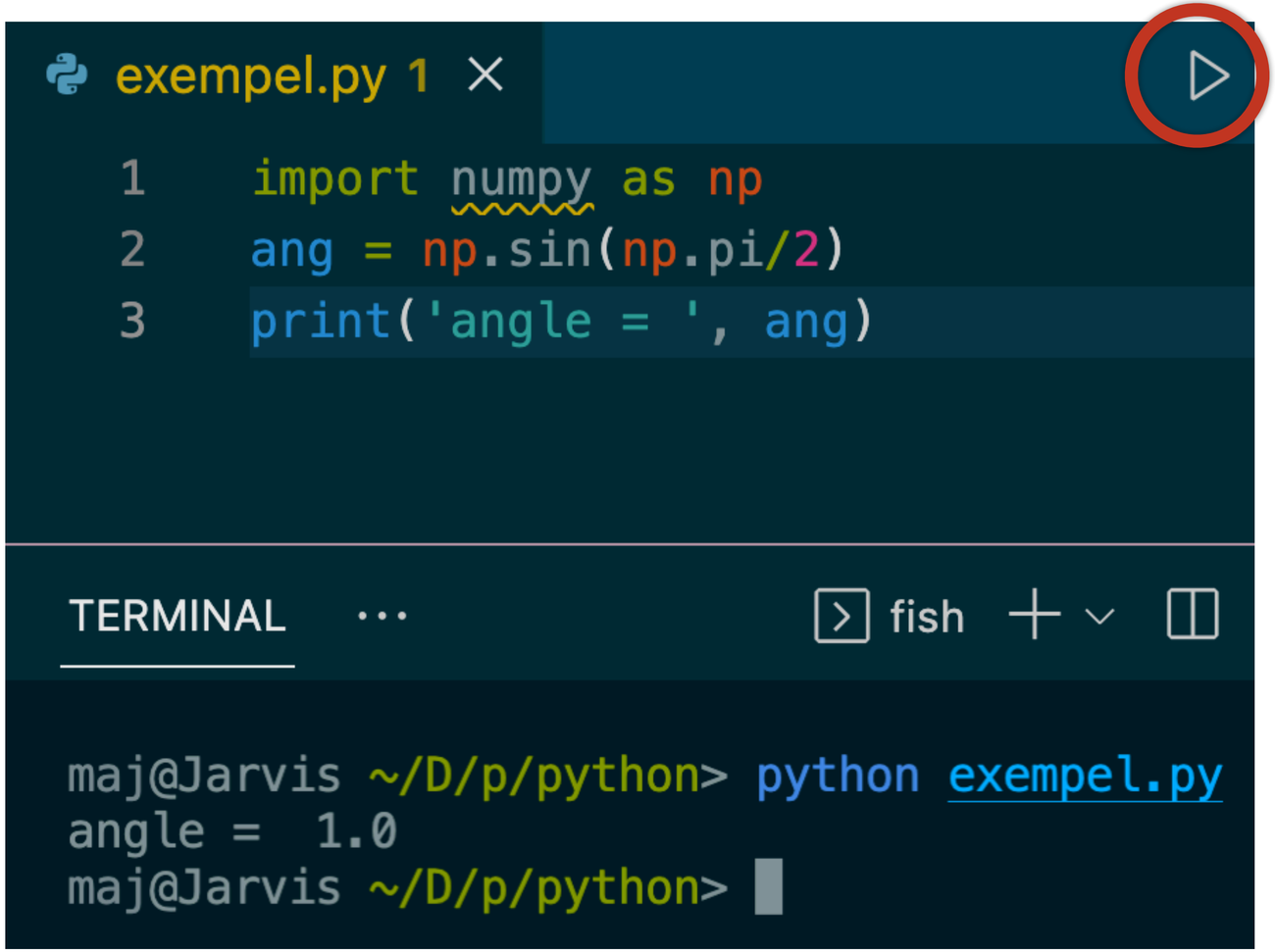
Python repl

* Pythonprogram exekveras:
  - rad för rad i en repl (read-eval-print-loop) i terminalen. repl:en startas genom att skriva `python` i terminalen.
  - från en fil med kommandon (python filnamn.py)



Pythonscript

* Script används för att spara en sekvens av kommandon i ett program.
* Programmen sparas i en textfil med filändelsen ".py"
* Exekvera filen:
  - med pilen i VS Code.
  - eller genom att skriva "`python filnamnet`" i terminalen.



Komma igång med NumPy

* Numpy är ett pythonbibliotek som importeras så här:

```
import numpy as np
```

* Variabler och funktioner från biblioteket anropas med `np.<namn>`, t ex `np.pi` och `np.sin(0.5)`

* Typen för en variabel behöver inte deklareras, den härleds från sitt värde:

```
x = 1 #int
y = 1.0 #float
s = ‘hello’ #sträng
li = [1, 1, 2, 3, 5] #lista
```

Standardfunktioner **Python**, exempel

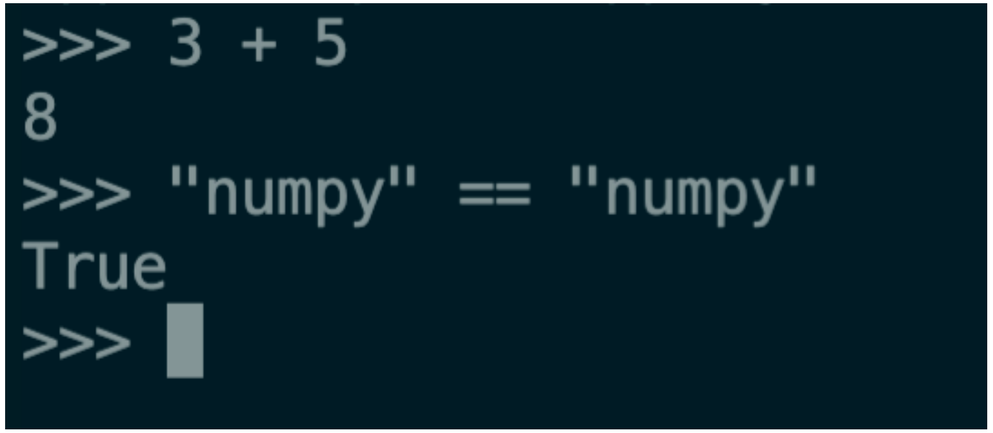
+, -, * vanliga matematiska operationer

/ ger ett flyttal 5 / 2 == 2.5

// heltalsdivision, 5 // 2 == 2

% är modulo (restdivision) 10 % 2 == 0

** potens, t ex 2 ** 3 == 8



== jämför (värde) likhet, returnerar True eller False (specifikt, den kolla innehållet i listors/arrayer)

round(x)

abs(z)

Standardfunktioner **NumPy**, exempel

```
>>> import numpy as np
>>> np.sin(np.pi/2)
1.0
>>>
```

np.round(x)

np.floor(x)

np.sin(x) #OBS! Radianer

np.cos(x)

np.tan(x)

np.sqrt(x) # för positiva tal

np.emath.sqrt(-3) #klarar negativa tal

np.power(x, y)

np.exp(x)

np.log(x)

np.log2(x) / np.log10(x)

Imaginära tal

Imaginära tal skrivs med j, t ex

1j, 2 + 2j, 3.0 + 4.5j

np.real(z) / np.imag(z)

eller: z.real / z.imag

np.abs(z) / np.angle(z)

np.conj(z)

```
>>> import numpy as np
>>> c = 1j
>>> c.real
0.0
>>> c.imag
1.0
>>>
```

Egendefinierade funktioner

* Vi kan definiera egna funktioner
* Enkla enradsfunktioner kan definieras som en **lambda**-funktion och lagras i en variabel:

```
f = lambda x: np.sqrt(1+np.sin(x))
g = lambda x,y: np.sin(x)+ y**2
```

* Funktionen kan sedan användas i uttryck:

```
z = f(3) + g(5,9)
```

* Skriv ut resultatet med `print('z = ', z)`

Egendefinierade funktioner forts.

* Vi kan också definiera vanliga **namngivna** funktioner:

```
def f(x):
    return np.sqrt(1+np.sin(x))

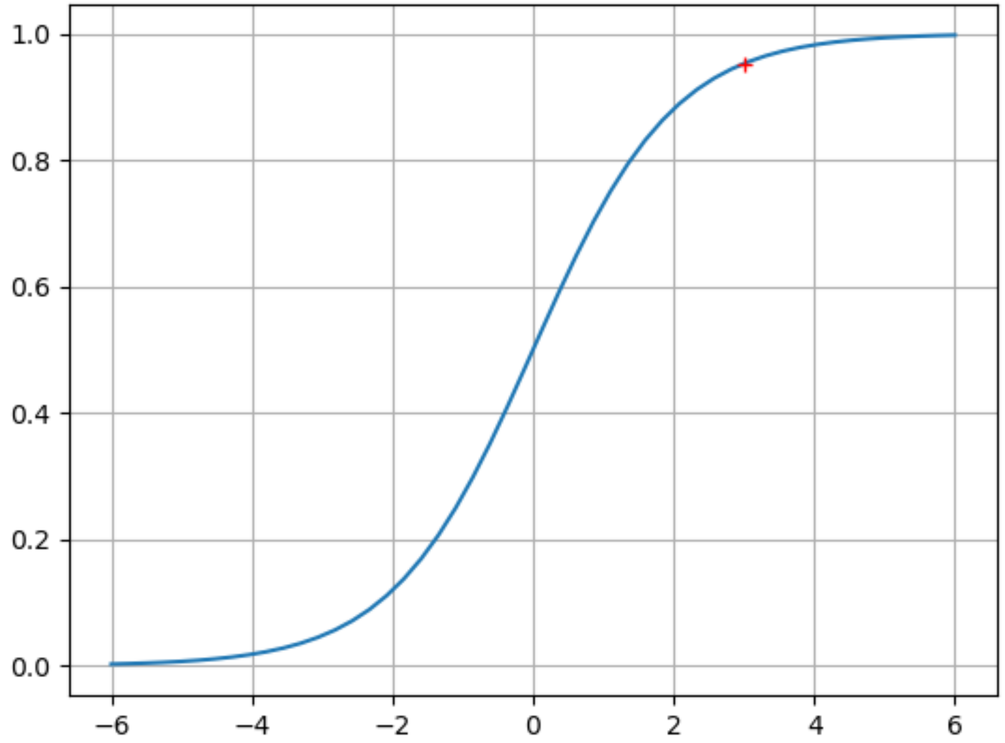
y = f(3)
```

* Tips! Funktioner kan refereras med variabler:
  - ```
    fcopy = f
    y = fcopy(3) # anropar funktionen f ovan
    ```
  - eller:
    ```
    import numpy
    np = numpy
    ```
    ```

Plotta grafer: inledande exempel

```
def sigma(x):
    return 1/(1 + np.exp(-x))

y = sigma(3) # 0.9525741
```



Funktionen kallas ofta *logistic curve* och är en populär *aktiveringsfunktion* i *neurala nätverk (mer om detta senare!).*

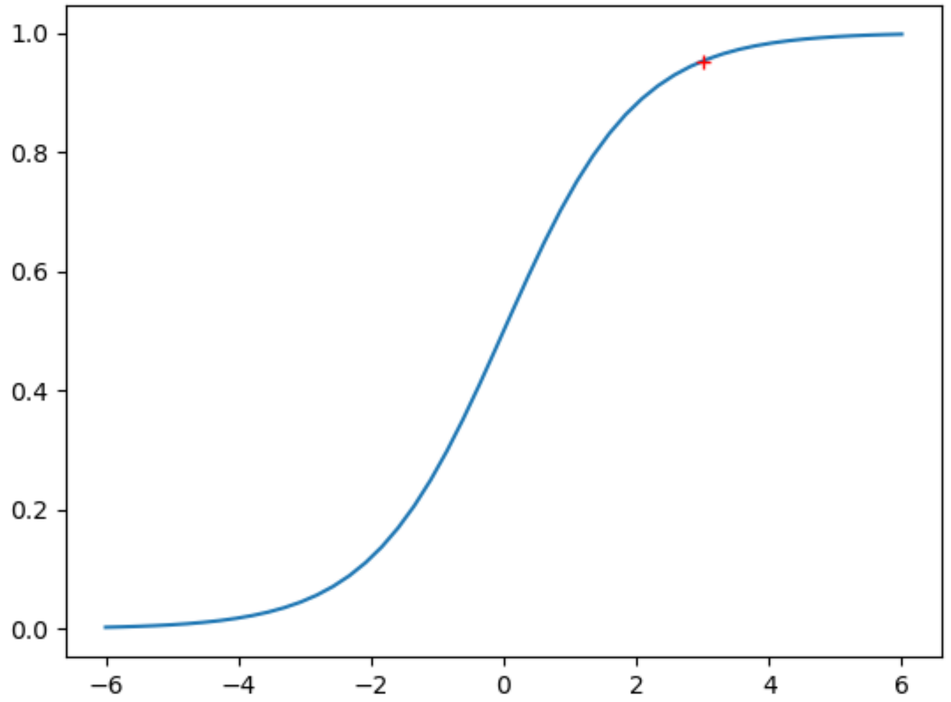
Följder av värden

* Ibland vill vi skapa en vektor med en följd av värden, t.ex.
  [1 2 3 4 5 6 7 8 9 10]

* **np.linspace**-funktionen ger ett antal jämnt fördelade värden i ett intervall
* `np.linspace(0,10,42)` ger 42 värden mellan 0 och 10
* Notera att argument kan namnges:
  `np.linspace(0,10, num = 42)`
* `np.linspace(0,10)` ger **50** värden mellan 0 och 10

Plotta grafer

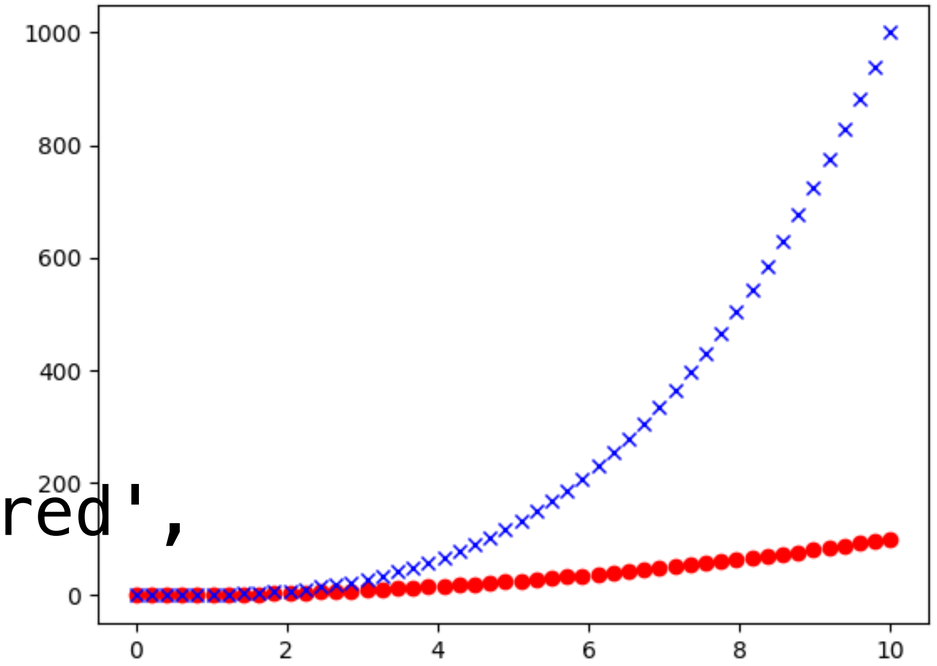
* För att plotta figurer använder vi paketet `matplotlib.pyplot`
* Vi importerar paketet och döper det till `plt`:
  `import matplotlib.pyplot as plt`
* `plt.plot(x, y)` plottar punkterna (x_i, y_i) och drar linjer mellan varandra följande punkter.
* Det går att plotta många grafer i samma bild.
* För att visa figuren använder vi `plt.show()` efter `plt.plot(…)`



Plotta grafer

* Det finns **många** valfria argument till plot, t ex

```
import numpy as np
import matplotlib.pyplot as plt
x = np.linspace(0, 10)
y = x **2
g = x **3
plt.plot(x, y, marker = 'o', color = 'red',
          linestyle = 'none')
plt.plot(x, g, ‘bx') #alternativ: b = blue, x för
markörsymbol
plt.show()
```



Se https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.plot.html

Vektorer och matriser

* Vektorer och matriser används för att hantera data, t ex
  * en vektor med mätdata med hastigheter från en bil
  * ett foto som representeras som en matris med pixlarnas RGB-värden (3 dimensioner: x-position, y-position och sen tre värden för RGB),
    t ex i biblioteket `opencv` för datorseende.

* Matriser används för att lösa många numeriska problem (t ex för att lösa ekvationssystem).

* Ofta är det stora mängder data och tunga beräkningar, NumPy har optimerat matris-beräkningar för att det ska gå snabbt 💪

Vektorer och matriser forts.

* Vektorer och matriser
* Skapa matriser
* Matrisberäkningar: addition, multiplikation, ...
* Sätta samman matriser
* Skapa matriser med `eye`, `ones`
* Lös ekvationssystem med matriser
* Elementvisa operationer

Vektorer och matriser forts.

* NumPy arbetar generellt med komplexa matriser
* Matriser är n-dimensionella arrayer av numpy-typen **ndarray**.
* En vektor är 1-dimensionell ndarray.

$$A = \begin{pmatrix} 1 & 1 & -1 \\ 2 & 1 & 1 \\ 4 & 3 & -1 \end{pmatrix}$$

Skapa matriser

* Vektorer skapas med en **lista** som argument:

```
v = np.array([2.0, 4.0])
c = np.array([2 + 1j, 4 + 2j])
```

* Matriser med en **lista med listor**:

```
A = np.array([[1, 1, -1], [2, 1, 1], [4, 3, -1]])
```

$$A = \begin{pmatrix} 1 & 1 & -1 \\ 2 & 1 & 1 \\ 4 & 3 & -1 \end{pmatrix}$$

Läsa/sätta värden i matriser

* `A = np.array([[1, 1, -1], [2, 1, 1], [4, 3, -1]])`
* Läsa enskilda element:

  ```
  A[0, 2] #element på rad 0 och kolumn 2, dvs -1
  ```
* Tilldelning:

  ```
  A[0, 2] = 5
  ```

Addera matriser

```
>>> A = np.array([[1, 1, -1], [2, 1, 1],
[4, 3, -1]])
```

$$A = \begin{pmatrix} 1 & 1 & -1 \\ 2 & 1 & 1 \\ 4 & 3 & -1 \end{pmatrix}$$

```
>>> B = np.array([[8, 3, 10], [-5, -2,
-7], [4, 11, 22]])
```

$$B = \begin{pmatrix} 8 & 3 & 10 \\ -5 & -2 & -7 \\ 4 & 11 & 22 \end{pmatrix}$$

Två matriser adderas elementvis: $C = \begin{pmatrix} 9 & 4 & 9 \\ -3 & -1 & -6 \\ 8 & 14 & 21 \end{pmatrix}$

```
>>> C = A + B
```

Addera matriser

Det går också bra att addera en skalär till matrisen (adderas till alla element)

$$C = \begin{pmatrix} 9 & 4 & 9 \\ -3 & -1 & -6 \\ 8 & 14 & 21 \end{pmatrix}$$

```
>>> D = C + 5
```

$$D = \begin{pmatrix} 14 & 9 & 14 \\ 2 & 4 & -1 \\ 13 & 19 & 26 \end{pmatrix}$$

Vad händer när vi adderar matriser med olika dimensioner?

$$\begin{pmatrix} 9 & 4 & 9 \\ -3 & -1 & -6 \\ 8 & 14 & 21 \end{pmatrix} + \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix}$$

```
>>> E = C + np.array([[1, 2], [3, 4], [5, 6]])
```

Subtrahera matriser

Det går också bra att subtrahera en matris eller

en skalär till en matrisen (görs elementvis)

```
>>> F = A - B
```

och

```
>>> D = C - 5
```

$$\begin{pmatrix} 1 & 1 & -1 \\ 2 & 1 & 1 \\ 4 & 3 & -1 \end{pmatrix} - \begin{pmatrix} 8 & 3 & 10 \\ -5 & -2 & -7 \\ 4 & 11 & 22 \end{pmatrix} = \begin{pmatrix} -7 & -2 & -11 \\ 7 & 3 & 8 \\ 0 & -8 & -23 \end{pmatrix}$$

$$D = \begin{pmatrix} 9 & 4 & 9 \\ -3 & -1 & -6 \\ 8 & 14 & 21 \end{pmatrix} - 5 = \begin{pmatrix} 14 & 9 & 14 \\ 2 & 4 & -1 \\ 13 & 19 & 26 \end{pmatrix}$$

Multiplicera matriser

- Multiplikation med skalär multiplicerar elementvis:

```
>>> 3 * A
```

$$3 * \begin{pmatrix} 1 & 1 & -1 \\ 2 & 1 & 1 \\ 4 & 3 & -1 \end{pmatrix} = \begin{pmatrix} 3 & 3 & -3 \\ 6 & 3 & 3 \\ 12 & 9 & -3 \end{pmatrix}$$

- Matrismultiplikation använder **np.matmul**

```
>>> C = np.matmul(A, B)
```

$$\begin{pmatrix} 1 & 1 \\ 2 & 2 \end{pmatrix} * \begin{pmatrix} 1 & 2 \\ 1 & 2 \end{pmatrix} = \begin{pmatrix} 2 & 4 \\ 4 & 8 \end{pmatrix}$$

A B C

Raderna multipliceras med kolumnerna, dvs första raden i A multiplicerat med första kolumnen i B blir värdet på plats $C_{1,1}$: 1 * 1 + 1 * 1 = 2

Multiplicera matriser forts.

* OBS! OBS! OBS! Använd inte *!
  Multiplicationstecknet * gör en **elementvis** multiplikation
  som beror på dimensionerna av arrayerna
  >>> C = A * B

$$(1 \quad 2) * (1 \quad 3) = (1 \quad 6)$$

A B C

* **På ibland oväntade sätt**!
  >>> C = A * B

$$(1 \quad 2) * \begin{pmatrix} 1 & 3 \\ 2 & 4 \end{pmatrix} = ????$$

A B C

(Här multipliceras varje rad i B med [1, 2])

Multiplicera matriser forts.

* OBS! OBS! OBS! Använd inte *!
  Multiplicationstecknet * gör en **elementvis** multiplikation
  som beror på dimensionerna av arrayerna
  >>> C = A * B

$$(1 \quad 2) * (1 \quad 3) = (1 \quad 6)$$

A B C

* **På ibland oväntade sätt**!
  >>> C = A * B

$$(1 \quad 2) * \begin{pmatrix} 1 & 3 \\ 2 & 4 \end{pmatrix} = \begin{pmatrix} 1 & 6 \\ 2 & 8 \end{pmatrix}$$

A B C

(Här multipliceras varje rad i B med [1, 2])

Sätta samman matriser

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \qquad B = \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix}$$

* A och B kan sättas samman antingen vertikalt som nya rader eller horisontellt som nya kolumner.

$$C = \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \\ 7 & 8 \end{pmatrix} \qquad D = \begin{pmatrix} 1 & 2 & 5 & 6 \\ 3 & 4 & 7 & 8 \end{pmatrix}$$

```
C = np.vstack([A, B]) #sätter ihop dem vertikalt
D = np.hstack([A, B]) #sätter ihop dem horisontellt.
Notera att argumentet är en lista med matriser!
```

Sätta samman matriser forts.

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \qquad B = \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix} \qquad C = \begin{pmatrix} 9 & 10 \\ 11 & 12 \end{pmatrix}$$

* Det går bra att sätta ihop fler matriser:

```
D = np.hstack([A, B, C])
```

$$D = \begin{pmatrix} 1 & 2 & 5 & 6 & 9 & 10 \\ 3 & 4 & 7 & 8 & 11 & 12 \end{pmatrix}$$

eye och ones

Det finns funktioner som skapar vissa standardmatriser:

```
I = np.eye(5)

T = np.eye(3) * 2

M = np.ones((3, 2)) * 4

D = np.eye(4, k = -1) * 3
```

$$T = \begin{pmatrix} 2.0 & 0.0 & 0.0 \\ 0.0 & 2.0 & 0.0 \\ 0.0 & 0.0 & 2.0 \end{pmatrix}$$

$$M = \begin{pmatrix} 4.0 & 4.0 \\ 4.0 & 4.0 \\ 4.0 & 4.0 \end{pmatrix}$$

$$I = \begin{pmatrix} 1.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 1.0 \end{pmatrix}$$

$$D = \begin{pmatrix} 0.0 & 0.0 & 0.0 & 0.0 \\ 3.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 3.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 3.0 & 0.0 \end{pmatrix}$$

Övningsuppgifter:

* Beräkna $A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} + \begin{pmatrix} 2 & 3 \\ 4 & 5 \end{pmatrix}$

* Skapa matrisen $\begin{pmatrix} 1j & 0 & 0 & 4 & 4 \\ 0 & 1j & 0 & 4 & 4 \\ 0 & 0 & 1j & 4 & 4 \end{pmatrix}$

Elementvisa operationer

* Numpys standardoperationer

  np.sin, **np.log** etc kan hantera numpy-arrayer som argument (notera även potenser **)!

  T ex

```
f = lambda x: np.sqrt(x)
g = lambda x,y: np.sin(x)+ y**2
```

* Funktionen kan sedan användas i uttryck:

```
v = np.array([1, 4, 9])
z = f(v) #[1, 2, 3]
x = np.array([np.pi, np.pi/2, 0])
```

y = g(x, v) #[0, 1, 0] + [1, 16, 81] = [1, 17, 81]

Ekvationssystem

$$3x_1 + 2x_2 - x_3 = 1$$

$$2x_1 - 2x_2 + 4x_3 = -2$$

$$-x_1 + x_2/2 - x_3 = 0$$

Lösa ekvationssystem

$3x_1 + 2x_2 - x_3 = 1$

$2x_1 - 2x_2 + 4x_3 = -2$

$-x_1 + x_2/2 - x_3 = 0$

$$\begin{pmatrix} 3 & 2 & -1 \\ 2 & -2 & 4 \\ -1 & 1/2 & -1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 1 \\ -2 \\ 0 \end{pmatrix}$$

A x B

* Lösningen kan beräknas med linjär algebra, genom att invertera matrisen och multiplicera båda sidorna med inversen …

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 3 & 2 & -1 \\ 2 & -2 & 4 \\ -1 & 1/2 & -1 \end{pmatrix}^{-1} \begin{pmatrix} 1 \\ -2 \\ 0 \end{pmatrix}$$

x A^{-1} B

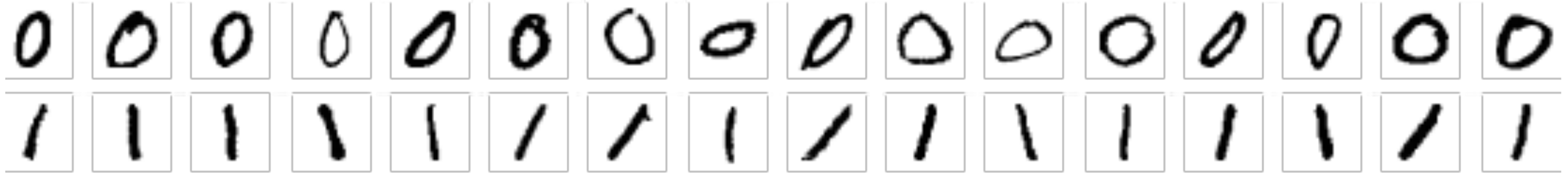
eller med **x = np.linalg.solve(A, B)**

Plotta grafer: övning

Plotta följande funktion i x-intervallet 0 till 10:

$$f(x) = e^{-x/4} \cdot \sin(2\pi x)$$

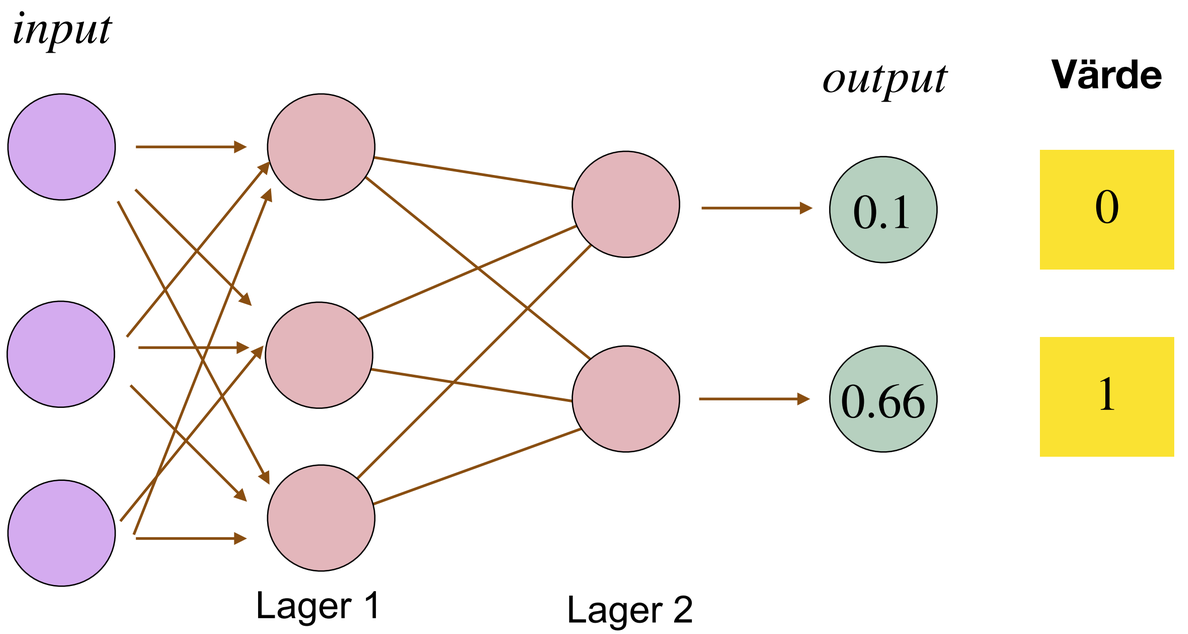
Exempel: Neurala nätverk



* Vi vill känna igen handskrivna siffror (bara 0 och 1 för enkelhetens skull).
* Vi har en massa bilder med 28 x 28 pixlar i gråskala 0 (svart) och 255 (vitt). Så varje bild har 784 tal.
* Vi vill automatiskt känna igen vilken siffra det är.

Exempel: Neurala nätverk

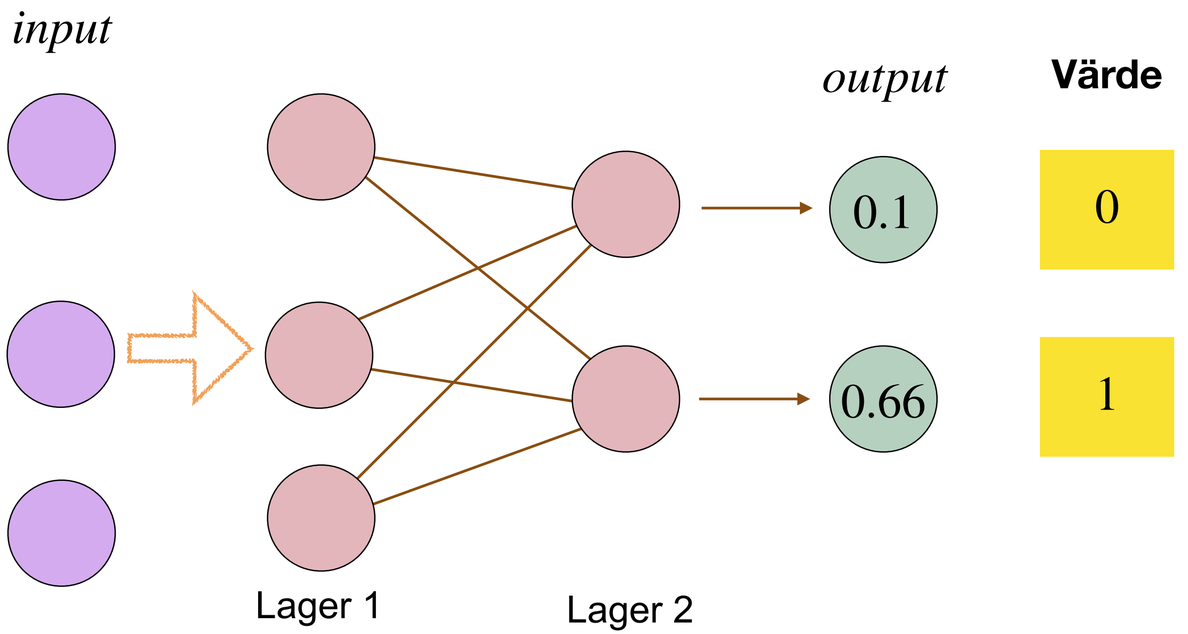
* Ett neuralt nätverk är en graf med noder som kallas neuron.
* Kanterna mellan noderna har vikter.
* Beräkningarna görs från vänster till höger.
* Vi kan ha flera *lager* i vårt nätverk.



Mini nätverk

Exempel: Neurala nätverk

* Ett neuralt nätverk är en graf med noder som kallas neuron.
* Kanterna mellan noderna har vikter.
* Beräkningarna görs från vänster till höger.
* Vi kan ha flera *lager* i vårt nätverk.



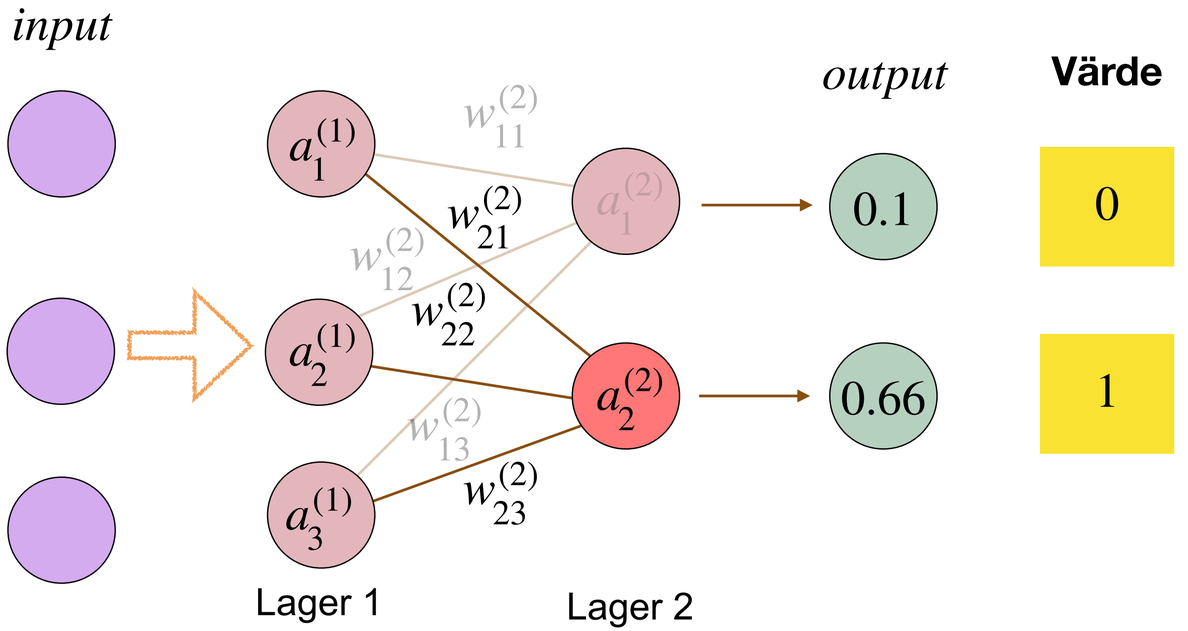
Mini nätverk

Exempel: Neurala nätverk

* Varje nod ger ut ett värde

$a_k^{(L)}$ ← Numret för lagret
← Numret för noden

$w_{\text{till, från}}$



* Värdet för en nod beräkans:

1. ta den viktade summan av alla inkommande värden plus en konstant:

$$z_2^{(2)} = w_{21}a_1^{(1)} + w_{22}a_2^{(1)} + w_{23}a_3^{(1)} + b_2^{(2)}$$

2. anropar sen en *aktiveringsfunktion*

$$a_2^{(2)} = \sigma(z_2^{(2)})$$

Mini nätverk

Exempel: Neurala nätverk

* Varje nod ger ut ett värde

$a_k^{(L)}$ ← Numret för lagret
← Numret för noden

$w_{\text{till, från}}$

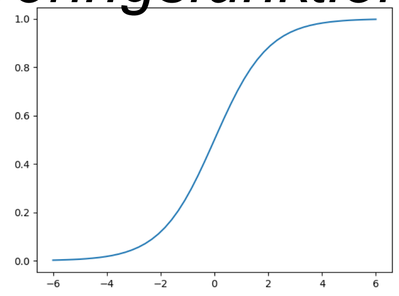
* Värdet för en nod beräknas:

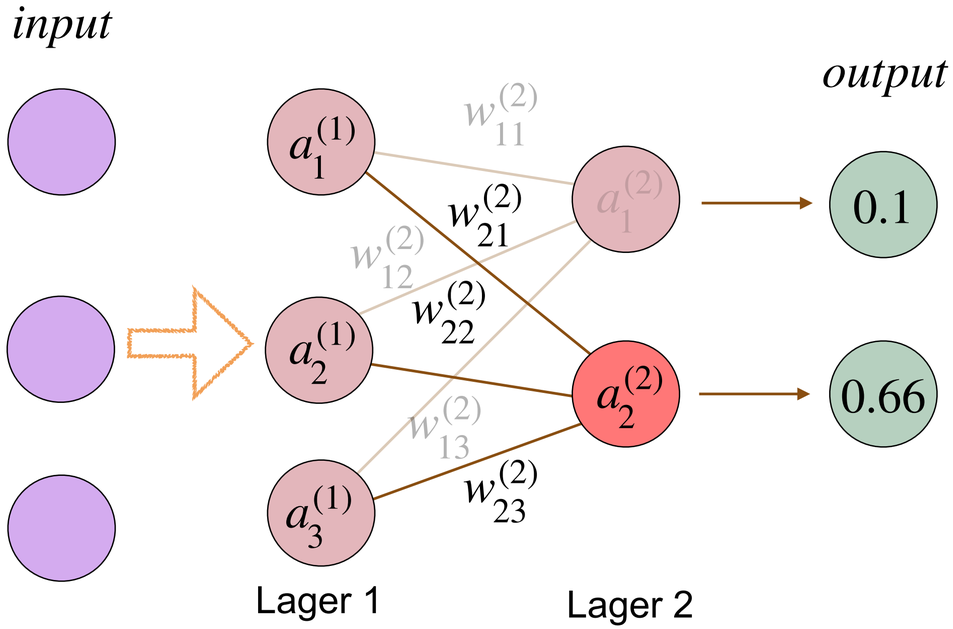
1. ta den viktade summan av alla inkommande värden plus en konstant:

$$z_2^{(2)} = w_{21}a_1^{(1)} + w_{22}a_2^{(1)} + w_{23}a_3^{(1)} + b_2^{(2)}$$

2. anropa sen en *aktiveringsfunktion*

$$a_2^{(2)} = \sigma(z_2^{(2)})$$





Mini nätverk

Exempel: Neurala nätverk

* Varje nod ger ut ett värde

$a_k^{(L)}$ ← Numret för lagret
← Numret för noden

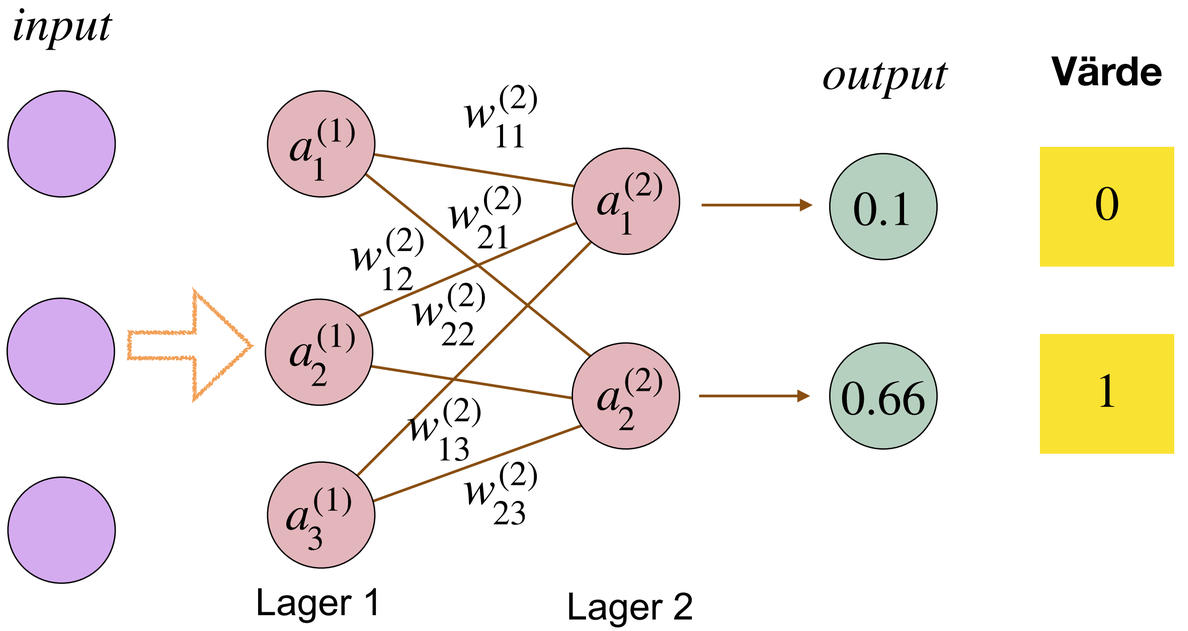
$w_{\text{till, från}}$

* Värdet för en nod beräknas:
  1 ta den viktade summan av alla inkommande värden plus en konstant:

$$z_x^{(2)} = \sum_i w_{xi} a_i^{(1)} + b_x^{(2)}$$

* anropa en *aktiveringsfunktion*

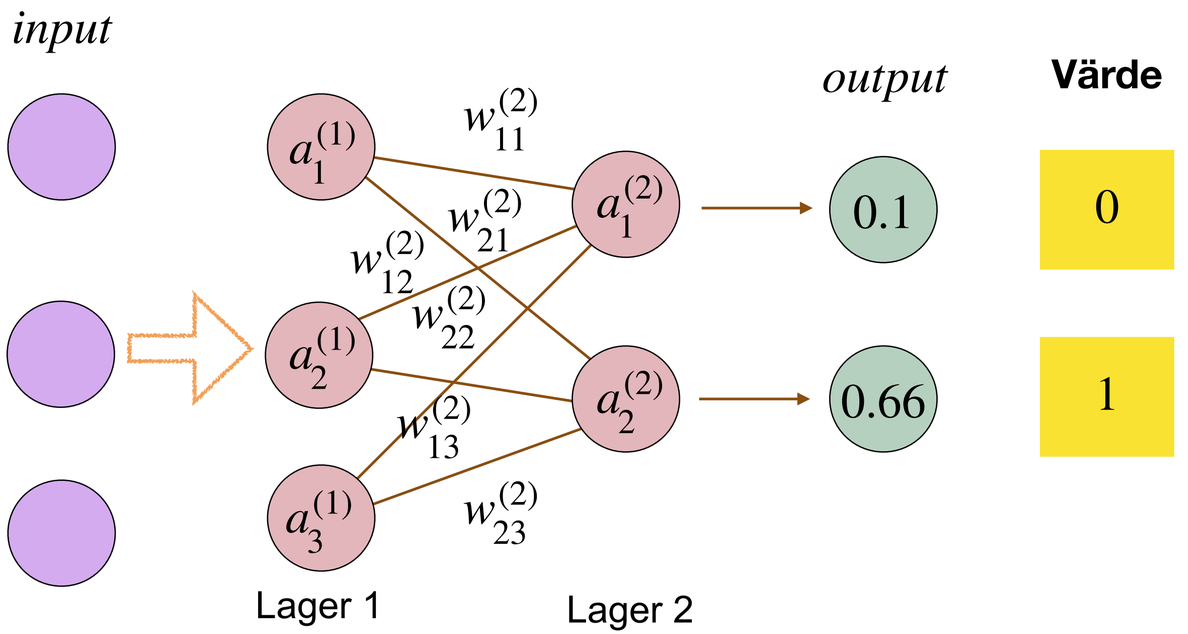
$$a_x^{(L)} = \sigma(z_x^{(L)})$$



Mini nätverk

Exempel: Neurala nätverk

* $z_x^{(2)} = \sum_i w_{xi} a_i^{(1)} + b_x^{(2)}$
* *Aktiveringsfunktion* $a_x^{(L)} = \sigma(z_x^{(L)})$



* Detta kan skrivas med hjälp av matriser: $Z = WA + B$

$$\begin{pmatrix} z_1^{(2)} \\ z_2^{(2)} \end{pmatrix} = \begin{pmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{pmatrix} \begin{pmatrix} a_1^{(1)} \\ a_2^{(1)} \\ a_3^{(1)} \end{pmatrix} + \begin{pmatrix} b_1^{(2)} \\ b_2^{(2)} \end{pmatrix}$$

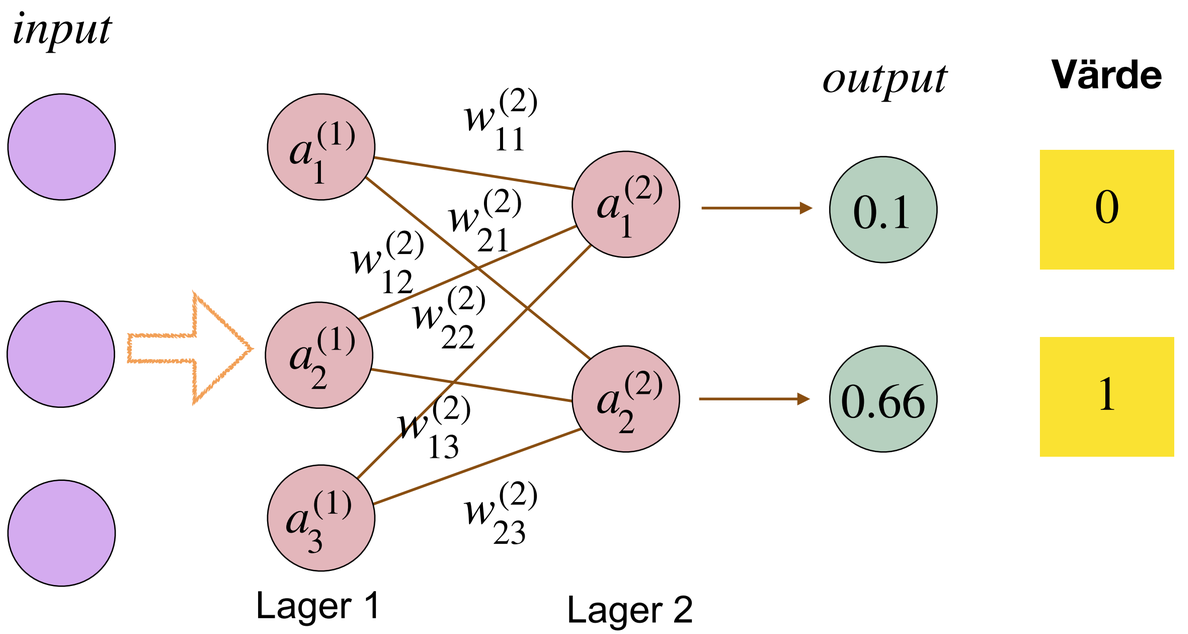
Mini nätverk

och $A_{\text{ut från lagret}} = \sigma(Z)$

Exempel: Neurala nätverk

För varje lager räknar vi ut värdena $Z = WA + B$

och $A_{\text{ut från lagret}} = \sigma(Z)$



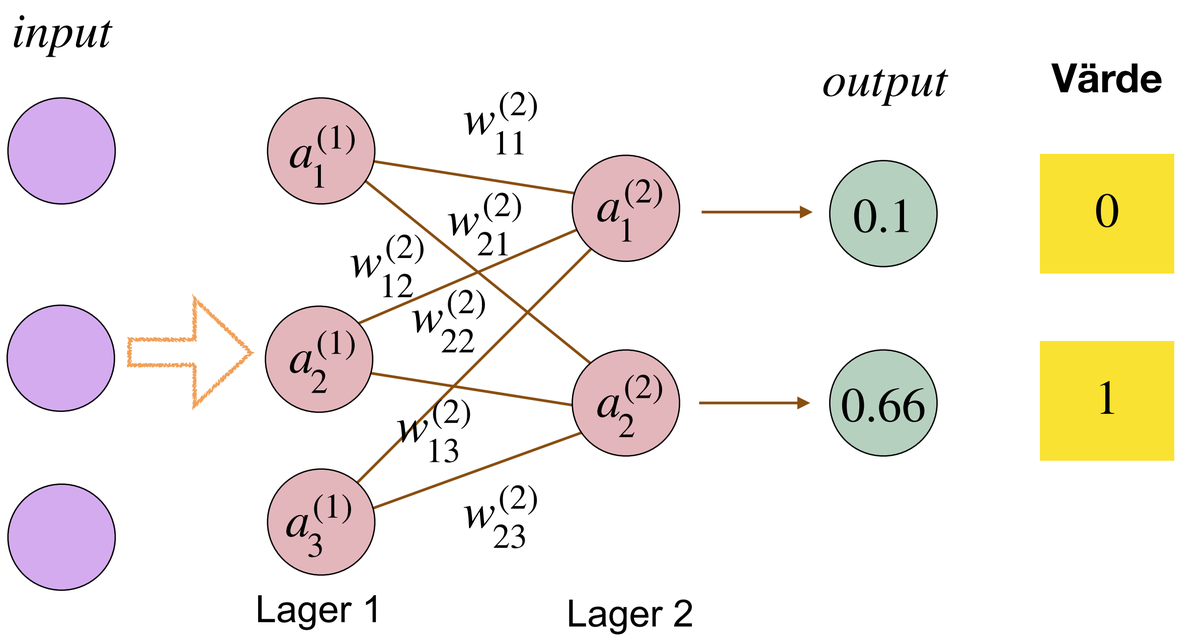
* Så när vi får en bild som input går vi genom varje lager i nätverket och räknar ut värdena med formlerna ovan. Det output som är högst är det talet vi tror det är.

Mini nätverk

Exempel: Neurala nätverk

För varje lager räknar vi ut värdena $Z = WA + B$

och $A_{\text{ut från lagret}} = \sigma(Z)$



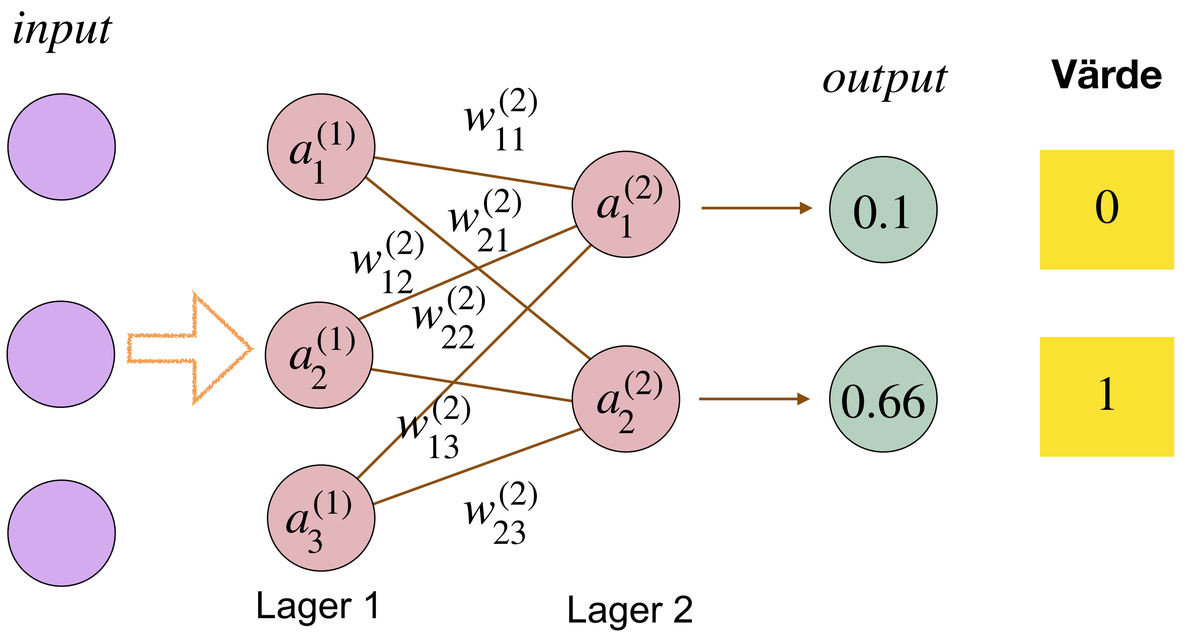
* Det som är svårt är att veta vad vikterna ska vara! Vi kan *träna* vårt nätverk med exempel och justera vikterna så att värdena går åt rätt håll.

Mini nätverk

Exempel: Neurala nätverk

För varje lager räknar vi ut värdena $Z = WA + B$

och $A_{\text{ut från lagret}} = \sigma(Z)$



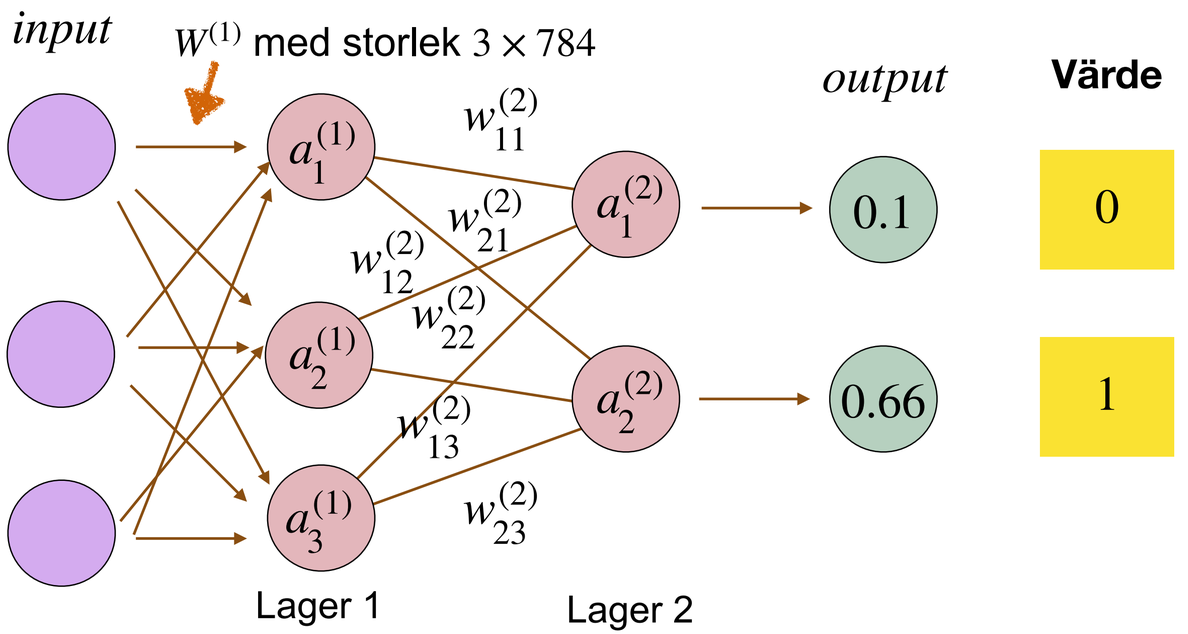
* Vi gör om bilden till en vektor med 784 tal.

Mini nätverk

Exempel: Neurala nätverk

För varje lager räknar vi ut värdena $Z = WA + B$

och $A_{\text{ut från lagret}} = \sigma(Z)$



* Vi gör om bilden till en vektor med 784 tal.

Mini nätverk

Inläsning av värden

* inläsning av en sträng:
  * `x = input()`
  * `x = input('Ange ett värde:’)`
  * `input` returnerar en sträng
  * typomvandla med `int(x)` eller `float(x)`

Exempel 1

```
x = input('Ange ett värde:')
x = int(x)
```

Övning

* Fibonaccis talserie börjar så här:
  0, 1, 1, 2, 3, 5, 8, 13, 21
* Serien kan beskrivas

$F_n = F_{n-1} + F_{n-2} \quad (n > 1)$

$F_0 = 0, \quad F_1 = 1$

* Hur kan vi skriva ett Pythonscript som läser in ett tal n (där n > 1) och skriver ut F_n?

Tips och tricks

* `True` har värdet 1 och `False` värdet 0

```
>>> True == 1
True
>>> True * 2
2
>>> 5//3 + (5 % 3 > 0) # avrundar divisionen uppåt
```

* Tupler är icke-muterbara listor av värden (som kan ha olika typ)

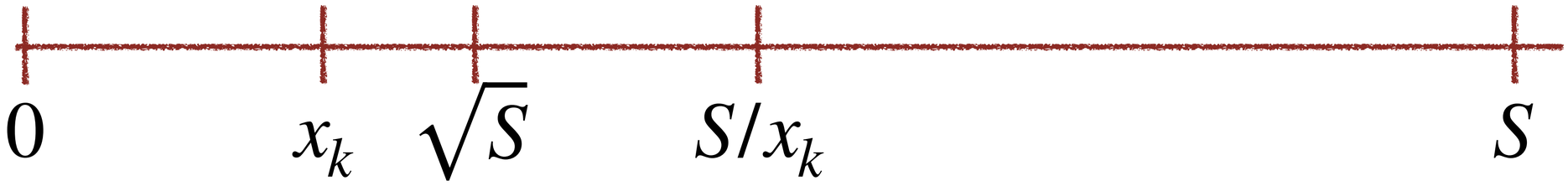
```
eevi = (“Eevi”, 19)
eevi[0] # “Eevi”
name, age = eevi
```

* Tupler kan användas för att returnera många värden från en funktion.

Temauppgift labb 1

* Approximera kvadratroten av ett tal $x^2 = S$

$$x_{k+1} = \frac{x_k + (S/x_k)}{2}$$ och vi börjar med $x_0 = S/2$



* Efter många steg kommer blir värdet x, dvs $x_\infty = S/x_\infty$

Sammanfattning

* Att köra Pythonprogram
  * repl och script
* Typer, aritmetiska operationer och imaginära tal
* Standard funktioner, lambda och `def`
* Vektorer och matriser
  * Att skapa matriser, ändra värden
  * Operationer på matriser
  * Sätta samman matriser
  * Lösa ekvationssystem
* Lite om plottning
  * `linspace`
  * färger
* Pythonsyntax
  * `input()`