NTNU

Kunnskap for en bedre verden

TDT4110 Informasjonsteknologi grunnkurs:
Python: Repetisjon
Matriser (2D-lister)
try…except
rekursjon
skrive pent til skjerm

Terje Rydland - IDI/NTNU

Læringsmål og pensum

- Mål
  - Repetere pensum ved å se på eksamensoppgaver
- Pensum: Starting out with Python
  - 3rd edition: Kapittel 1-9 + kapittel 12
  - 2nd edition: Kapittel 1-10 + kapittel 13

To-dimensjonale lister

- To-dimensjonale liste: liste som inneholder andre lister som elementer
  - Også kjent som nøstede lister
  - Vanlig å betrakte to-dimensjonale lister som om de har rekker og kolonner
  - Nyttig til å jobbe med flere datasett
- For å prosessere data i to-dimensjonale lister trenger man å bruke indekser
- Typisk brukes nøstede løkker til å prosessere dem

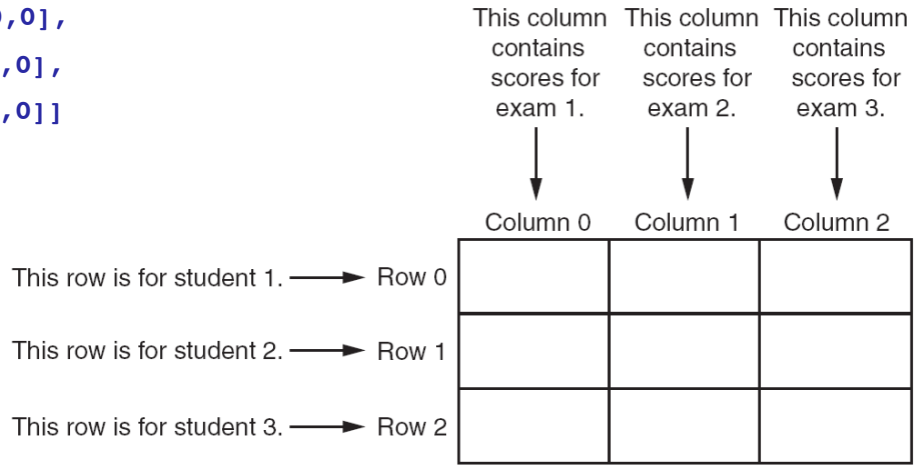
Lage to-dimensjonale lister (liste av lister)

```
students=[      ['Joe','Kim'],
                ['Sam','Sue'],
                ['Kelly','Chris']]
```

	Column 0	Column 1
Row 0	'Joe'	'Kim'
Row 1	'Sam'	'Sue'
Row 2	'Kelly'	'Chris'

Lage to-dimensjonale lister (lister av lister)

```
scores=[[0,0,0],
        [0,0,0],
        [0,0,0]]
```



Hente ut verdier fra to-dimensjonale lister:

	Column 0	Column 1	Column 2
Row 0	scores[0][0]	scores[0][1]	scores[0][2]
Row 1	scores[1][0]	scores[1][1]	scores[1][2]
Row 2	scores[2][0]	scores[2][1]	scores[2][2]

```
print(scores[1][1])     # Skriver ut element (1,1)
x=scores[2][0]          # Setter x lik element (2,0)
scores[0][0] = 9        # Setter element (0,0) lik 9
```

Lage to-dimensjonale tabeller av vilkårlig størrelse

- Man kan også opprette en fler-dimensjonal tabell av en gitt størrelser uten å angi alle elementene:
  - Lage en 2-dimensjonal 10x10 matrise med 0er:

```
tabell_10x10 = [[0 for col in range(10)]
                   for row in range(10)]
```

  - Lage en 3-dimensjonal 3x3x3 matrise med 1ere:

```
tabell_3d = [[[1 for x in range(3)]
                 for y in range(3)]
                 for z in range(3)]
```


Eksempel

util.py - /Users/terjery/Library/Mobile Documents/com~apple~CloudDocs/ITGK 2016/Python/1Forelesninger/Uke 47/Matriser/util.py (3.5.2)

```
import random

def fyllMatrise(matrise):
    fyll = 1
    for rad in range(len(matrise)):
        for kol in range(len(matrise[0])):
            matrise[rad][kol] = fyll
            fyll += 1
    return matrise

def transponer(matrise):
    trans = [[0 for kol in range(len(matrise))] for rad in range(len(matrise[0]))]
    for rad in range(len(matrise)):
        for kol in range(len(matrise[0])):
            trans[kol][rad] = matrise[rad][kol]
    return trans

def dimensjon(a,b):
    return (len(a)==len(b)) and (len(a[0]) == len(b[0]))

def adder(a,b):
    if dimensjon(a,b):
        summ = [[0 for kol in range(len(a[0]))] for rad in range(len(a))]
        for rad in range(len(a)):
            for kol in range(len(a[0])):
                summ[rad][kol] = a[rad][kol] + b[rad][kol]
        return summ
    else:
        return 'Matrisene har ikke samme størrelse, og kan derfor ikke adderes'

def skrivPenMatrise(a):
    for rad in range(len(a)):
        for kol in range(len(a[rad])):
            print(str(a[rad][kol]).rjust(5),end='')
        print()
```

Ln: 36 Col: 0

Eksempel

util.py - /Users/terjery/Library/Mobile Documents/com~apple~CloudDocs/ITGK 2016/Python/1Forelesninger/Uke 47/Matriser/util.py (3.5.2)

```
import random

def fyllMatrise(matrise):
    fyll = 1
    for rad in range(len(matrise)):
        for kol in range(len(matrise[0])):
            matrise[rad][kol] = fyll
            fyll += 1
    return matrise
```

- Fyller matrisen med 1 2 3 4 5 6 7 8 9…
- Ytre for tar rad for rad
- Indre for tar alle kolonnene i en rad

Inn

1	1	1
1	1	1
1	1	1
1	1	1
1	1	1

Ut

1	2	3
4	5	6
7	8	9
10	11	12
13	14	15

Eksempel

```
def transponer(matrise):
    trans = [[0 for kol in range(len(matrise))] for rad in range(len(matrise[0]))]
    for rad in range(len(matrise)):
        for kol in range(len(matrise[0])):
            trans[kol][rad] = matrise[rad][kol]
    return trans
```

- Legger rad i `matrise` inn i kolonne i `trans`
  - `trans` skal ha like mange kolonner som `matrise` har rader, og like mange rader som `matrise` har kolonner
- Ytre for tar rad for rad i `matrise`
- Indre for tar alle kolonnene i en rad i `matrise`
  - `trans[kol][rad] = matrise[rad][kol]`

Inn

1	2	3
4	5	6
7	8	9
10	11	12
13	14	15

Ut

1	4	7	10	13
2	5	8	11	14
3	6	9	12	15

Eksempel

```
def dimensjon(a,b):
    return (len(a)==len(b)) and (len(a[0]) == len(b[0]))
```

- Sjekker om to matriser har samme dimensjon
  - Har de samme antall rader og samme antall kolonner

a

1	2	3
4	5	6
7	8	9
10	11	12
13	14	15

b

1	4	7	10	13
2	5	8	11	14
3	6	9	12	15

Resultat: `False`

Eksempel

```
def adder(a,b):
    if dimensjon(a,b):
        summ = [[0 for kol in range(len(a[0]))] for rad in range(len(a))]
        for rad in range(len(a)):
            for kol in range(len(a[0])):
                summ[rad][kol] = a[rad][kol] + b[rad][kol]
        return summ
    else:
        return 'Matrisene har ikke samme størrelse, og kan derfor ikke adderes'
```

- Legger sammen to matriser
  - Sjekker først om de har samme dimensjon (ellers kan de ikke adderes)
  - Nullstille målmatrisen
  - Går gjennom, rad for rad, kolonne for kolonne og adderer elementene

a

1	2	3
4	5	6
7	8	9
10	11	12
13	14	15

b

1	2	3
4	5	6
7	8	9
10	11	12
13	14	15

summ

2	4	6
8	10	12
14	16	18
20	22	24
26	28	30

Eksempel

```
def skrivPenMatrise(a):
    for rad in range(len(a)):
        for kol in range(len(a[rad])):
            print(str(a[rad][kol]).rjust(5),end='')
        print()
```

Ln: 36 Col: 0

- Skriver matrisen ut pent
  - Går gjennom rad for rad og linje for linje
  - Konverterer elementene til str og høyrejusterer med metoden .rjust(5)

a

1	2	3
4	5	6
7	8	9
10	11	12
13	14	15

```
    1    2    3
    4    5    6
    7    8    9
   10   11   12
   13   14   15
```


Oppgave 2 – Programmering (70 %)

I denne oppgaven skal du programmere metoder til et Snake spill, der spilleren skal styre en slange som beveger seg rundt på et spillbrett.

	0	1	2	3	4	5	6	7	8	9
0	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1
2	1	1	1	0	1	1	1	1	1	1
3	1	1	1	0	1	1	1	1	1	1
4	1	1	0	0	1	1	1	1	1	1
5	1	1	1	1	1	1	1	1	1	1
6	1	1	1	1	1	1	1	1	1	1
7	1	1	1	1	1	1	1	1	1	1
8	1	1	1	1	1	1	1	1	1	1
9	1	1	1	1	1	1	1	1	1	1

Spillbrettet en kvadratisk tabell av en vilkårlig størrelse, der 1 indikerer en tom rute.
Selve ormen representeres av to tabeller for hver akse (x og y) som tilsvarer lengden på ormen.
På figuren til venstre er ormen tegnet inn med tegnet '0' og er av lengde 5.
Tabellen for denne ormen er:

orm_x=[3,3,3,2,1]
orm_y=[2,3,4,4,4]

Hodet på ormen er markert med mørkest farge og vil da være punktet (3,2). Halen vil være punktet (1,4).

Merk at halen er tegnet inn som en 1'er, slik at ormen visker ut etter seg hvis den beveger seg på brettet.

	0	1	2	3	4	5	6	7	8	9
0	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1
2	1	1	1	0	1	1	1	1	1	1
3	1	1	1	0	1	1	1	1	1	1
4	1	1	0	0	1	1	1	1	1	1
5	1	1	1	1	1	1	1	1	1	1
6	1	1	1	1	1	1	1	1	1	1
7	1	1	1	1	1	1	1	1	1	1
8	1	1	1	1	1	1	1	1	1	1
9	1	1	1	1	1	1	1	1	1	1

Spillbrettet en kvadratisk tabell av en vilkårlig størrelse, der 1 indikerer en tom rute.
Selve ormen representeres av to tabeller for hver akse (x og y) som tilsvarer lengden på ormen.
På figuren til venstre er ormen tegnet inn med tegnet '0' og er av lengde 5.
Tabellen for denne ormen er:

orm_x=[3,3,3,2,1]
orm_y=[2,3,4,4,4]

Hodet på ormen er markert med mørkest farge og vil da være punktet (3,2). Halen vil være punktet (1,4).

a) (5%) Skriv funksjonen ***lag_tomt_brett*** som genererer (lager) et tomt kvadratisk spillbrett (tabell bestående av bare 1'ere av en gitt størrelse bestemt av en inn-parameter.

	0	1	2	3	4	5	6	7	8	9
0	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1
2	1	1	1	0	1	1	1	1	1	1
3	1	1	1	0	1	1	1	1	1	1
4	1	1	0	0	1	1	1	1	1	1
5	1	1	1	1	1	1	1	1	1	1
6	1	1	1	1	1	1	1	1	1	1
7	1	1	1	1	1	1	1	1	1	1
8	1	1	1	1	1	1	1	1	1	1
9	1	1	1	1	1	1	1	1	1	1

Spillbrettet en kvadratisk tabell av en vilkårlig størrelse, der 1 indikerer en tom rute.
Selve ormen representeres av to tabeller for hver akse (x og y) som tilsvarer lengden på ormen.
På figuren til venstre er ormen tegnet inn med tegnet '0' og er av lengde 5.
Tabellen for denne ormen er:

orm_x=[3,3,3,2,1]
orm_y=[2,3,4,4,4]

Hodet på ormen er markert med mørkest farge og vil da være punktet (3,2). Halen vil være punktet (1,4).

a) (5%) Skriv funksjonen ***lag_tomt_brett*** som genererer (lager) et tomt kvadratisk spillbrett (tabell bestående av bare 1'ere av en gitt størrelse bestemt av en inn-parameter.

```
def lag_tomt_brett(ruter):
    return [[1 for i in range(ruter)] for j in range(ruter)]
```

	0	1	2	3	4	5	6	7	8	9
0	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1
2	1	1	1	0	1	1	1	1	1	1
3	1	1	1	0	1	1	1	1	1	1
4	1	1	0	0	1	1	1	1	1	1
5	1	1	1	1	1	1	1	1	1	1
6	1	1	1	1	1	1	1	1	1	1
7	1	1	1	1	1	1	1	1	1	1
8	1	1	1	1	1	1	1	1	1	1
9	1	1	1	1	1	1	1	1	1	1

Spillbrettet en kvadratisk tabell av en vilkårlig størrelse, der 1 indikerer en tom rute.
Selve ormen representeres av to tabeller for hver akse (x og y) som tilsvarer lengden på ormen.
På figuren til venstre er ormen tegnet inn med tegnet '0' og er av lengde 5.
Tabellen for denne ormen er:

orm_x=[3,3,3,2,1]
orm_y=[2,3,4,4,4]

Hodet på ormen er markert med mørkest farge og vil da være punktet (3,2). Halen vil være punktet (1,4).

b) (5%) Skriv funksjonen ***legg_ut_epler*** som tar inn spillbrettet (en tabell av vilkårlig størrelse), samt antallet epler som skal legges ut vilkårlig på brettet. Epler blir representert av tallet 5.

	0	1	2	3	4	5	6	7	8	9
0	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1
2	1	1	1	0	1	1	1	1	1	1
3	1	1	1	0	1	1	1	1	1	1
4	1	1	0	0	1	1	1	1	1	1
5	1	1	1	1	1	1	1	1	1	1
6	1	1	1	1	1	1	1	1	1	1
7	1	1	1	1	1	1	1	1	1	1
8	1	1	1	1	1	1	1	1	1	1
9	1	1	1	1	1	1	1	1	1	1

Spillbrettet en kvadratisk tabell av en vilkårlig størrelse, der 1 indikerer en tom rute.
Selve ormen representeres av to tabeller for hver akse (x og y) som tilsvarer lengden på ormen.
På figuren til venstre er ormen tegnet inn med tegnet '0' og er av lengde 5.
Tabellen for denne ormen er:

orm_x=[3,3,3,2,1]
orm_y=[2,3,4,4,4]

Hodet på ormen er markert med mørkest farge og vil da være punktet (3,2). Halen vil være punktet (1,4).

b) (5%) Skriv funksjonen ***legg_ut_epler*** som tar inn spillbrettet (en tabell av vilkårlig størrelse), samt antallet epler som skal legges ut vilkårlig på brettet. Epler blir representert av tallet 5.


```
def legg_ut_epler(brett,antall):
    teller = 0
    while teller < antall:
        x = random.randint(0,len(brett)-1)
        y = random.randint(0,len(brett)-1)
        if brett[x][y] == 1:
            brett[x][y] = 5
            teller += 1
    return brett
```

	0	1	2	3	4	5	6	7	8	9
0	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1
2	1	1	1	0	1	1	1	1	1	1
3	1	1	1	0	1	1	1	1	1	1
4	1	1	0	0	1	1	1	1	1	1
5	1	1	1	1	1	1	1	1	1	1
6	1	1	1	1	1	1	1	1	1	1
7	1	1	1	1	1	1	1	1	1	1
8	1	1	1	1	1	1	1	1	1	1
9	1	1	1	1	1	1	1	1	1	1

Spillbrettet en kvadratisk tabell av en vilkårlig størrelse, der 1 indikerer en tom rute.
Selve ormen representeres av to tabeller for hver akse (x og y) som tilsvarer lengden på ormen.
På figuren til venstre er ormen tegnet inn med tegnet '0' og er av lengde 5. Tabellen for denne ormen er:

orm_x=[3,3,3,2,1]
orm_y=[2,3,4,4,4]

Hodet på ormen er markert med mørkest farge og vil da være punktet (3,2). Halen vil være punktet (1,4).

c) (5%) Skriv funksjonen ***sjekk_eple*** som tar inn en x- og y-posisjon (typisk hodet på ormen), samt spillbrettet. Funksjonen skal returnere True hvis et eple befinner seg på den angitte posisjonen (dvs. tallet 5), ellers False.

	0	1	2	3	4	5	6	7	8	9
0	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1
2	1	1	1	0	1	1	1	1	1	1
3	1	1	1	0	1	1	1	1	1	1
4	1	1	0	0	1	1	1	1	1	1
5	1	1	1	1	1	1	1	1	1	1
6	1	1	1	1	1	1	1	1	1	1
7	1	1	1	1	1	1	1	1	1	1
8	1	1	1	1	1	1	1	1	1	1
9	1	1	1	1	1	1	1	1	1	1

Spillbrettet en kvadratisk tabell av en vilkårlig størrelse, der 1 indikerer en tom rute.
Selve ormen representeres av to tabeller for hver akse (x og y) som tilsvarer lengden på ormen.
På figuren til venstre er ormen tegnet inn med tegnet '0' og er av lengde 5. Tabellen for denne ormen er:

orm_x=[3,3,3,2,1]
orm_y=[2,3,4,4,4]

Hodet på ormen er markert med mørkest farge og vil da være punktet (3,2). Halen vil være punktet (1,4).

c) (5%) Skriv funksjonen ***sjekk_eple*** som tar inn en x- og y-posisjon (typisk hodet på ormen), samt spillbrettet. Funksjonen skal returnere True hvis et eple befinner seg på den angitte posisjonen (dvs. tallet 5), ellers False.

```
def sjekk_eple(x,y,brett):
    return brett[x][y] == 5
```

	0	1	2	3	4	5	6	7	8	9
0	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1
2	1	1	1	0	1	1	1	1	1	1
3	1	1	1	0	1	1	1	1	1	1
4	1	1	0	0	1	1	1	1	1	1
5	1	1	1	1	1	1	1	1	1	1
6	1	1	1	1	1	1	1	1	1	1
7	1	1	1	1	1	1	1	1	1	1
8	1	1	1	1	1	1	1	1	1	1
9	1	1	1	1	1	1	1	1	1	1

Spillbrettet en kvadratisk tabell av en vilkårlig størrelse, der 1 indikerer en tom rute.
Selve ormen representeres av to tabeller for hver akse (x og y) som tilsvarer lengden på ormen.
På figuren til venstre er ormen tegnet inn med tegnet '0' og er av lengde 5.
Tabellen for denne ormen er:

orm_x=[3,3,3,2,1]
orm_y=[2,3,4,4,4]

Hodet på ormen er markert med mørkest farge og vil da være punktet (3,2). Halen vil være punktet (1,4).

d) (15%) Skriv funksjonen ***tegn_orm*** som tar inn spillbrettet (en tabell av vilkårlig størrelse) samt de to tabellene som representerer ormen (orm_x og orm_y) og tegner ormen inn i spillbrettet representert med 0'ere. Halen på ormen skal tegnes som en 1'er.

	0	1	2	3	4	5	6	7	8	9
0	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1
2	1	1	1	0	1	1	1	1	1	1
3	1	1	1	0	1	1	1	1	1	1
4	1	1	0	0	1	1	1	1	1	1
5	1	1	1	1	1	1	1	1	1	1
6	1	1	1	1	1	1	1	1	1	1
7	1	1	1	1	1	1	1	1	1	1
8	1	1	1	1	1	1	1	1	1	1
9	1	1	1	1	1	1	1	1	1	1

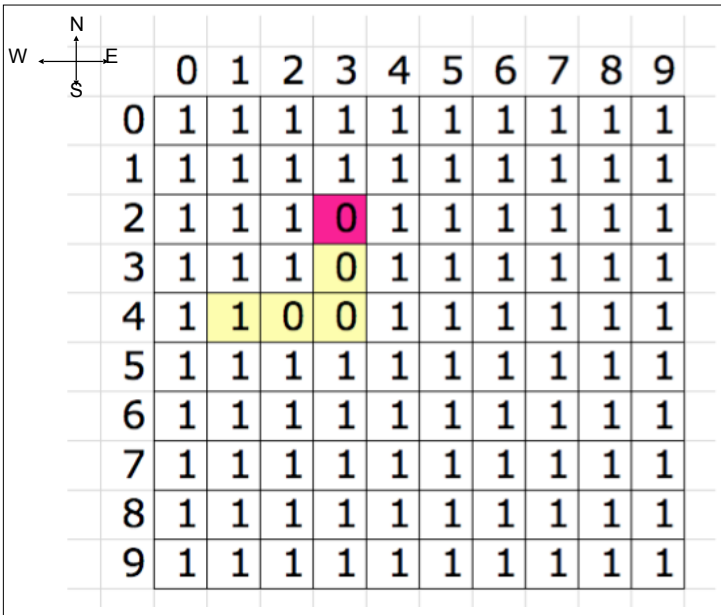
Spillbrettet en kvadratisk tabell av en vilkårlig størrelse, der 1 indikerer en tom rute.
Selve ormen representeres av to tabeller for hver akse (x og y) som tilsvarer lengden på ormen.
På figuren til venstre er ormen tegnet inn med tegnet '0' og er av lengde 5.
Tabellen for denne ormen er:

orm_x=[3,3,3,2,1]
orm_y=[2,3,4,4,4]

Hodet på ormen er markert med mørkest farge og vil da være punktet (3,2). Halen vil være punktet (1,4).

d) (15%) Skriv funksjonen ***tegn_orm*** som tar inn spillbrettet (en tabell av vilkårlig størrelse) samt de to tabellene som representerer ormen (orm_x og orm_y) og tegner ormen inn i spillbrettet representert med 0'ere. Halen på ormen skal tegnes som en 1'er.

```
def tegn_orm(brett,orm_x,orm_y):
    for x in range(len(orm_x)-1):
        brett[orm_y[x]][orm_x[x]] = 0
    return brett
```

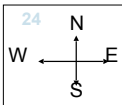


Spillbrettet en kvadratisk tabell av en vilkårlig størrelse, der 1 indikerer en tom rute.
Selve ormen representeres av to tabeller for hver akse (x og y) som tilsvarer lengden på ormen.
På figuren til venstre er ormen tegnet inn med tegnet '0' og er av lengde 5.
Tabellen for denne ormen er:

orm_x=[3,3,3,2,1]
orm_y=[2,3,4,4,4]

Hodet på ormen er markert med mørkest farge og vil da være punktet (3,2). Halen vil være punktet (1,4).

e) (15%) Skriv funksjonen ***sjekk_krasj*** som tar inn tabellene for ormen (orm_x og orm_y), samt retningen ormen skal bevege seg i (N, E, S eller W). Funksjonen skal sjekke om ormen krasjer i seg selv hvis den beveger seg i angitt retning. Funksjonen skal returnerer *True* hvis den krasjer og *False* hvis den ikke gjør det.



0 1 2 3 4 5 6 7 8 9
0 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1
2 1 1 1 0 1 1 1 1 1 1
3 1 1 1 0 1 1 1 1 1 1
4 1 1 0 0 1 1 1 1 1 1
5 1 1 1 1 1 1 1 1 1 1
6 1 1 1 1 1 1 1 1 1 1
7 1 1 1 1 1 1 1 1 1 1
8 1 1 1 1 1 1 1 1 1 1
9 1 1 1 1 1 1 1 1 1 1

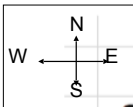
Spillbrettet en kvadratisk tabell av en vilkårlig størrelse, der 1 indikerer en tom rute.
Selve ormen representeres av to tabeller for hver akse (x og y) som tilsvarer lengden på ormen.
På figuren til venstre er ormen tegnet inn med tegnet '0' og er av lengde 5.
Tabellen for denne ormen er:

orm_x=[3,3,3,2,1]
orm_y=[2,3,4,4,4]

Hodet på ormen er markert med mørkest farge og vil da være punktet (3,2). Halen vil være punktet (1,4).

e) (15%) Skriv funksjonen ***sjekk_krasj*** som tar inn tabellene for ormen (orm_x og orm_y), samt retningen ormen skal bevege seg i (N, E, S eller W). Funksjonen skal sjekke om ormen krasjer i seg selv hvis den beveger seg i angitt retning. Funksjonen skal returnerer *True* hvis den krasjer og *False* hvis den ikke gjør det.

```
def sjekk_krasj(orm_x,orm_y,retning):
    x = orm_x[0]
    y = orm_y[0]
    sisteX = orm_x[len(orm_x)-1]
    sisteY = orm_y[len(orm_x)-1]
    if retning == 'E':
        x += 1
    elif retning == 'S':
        y += 1
    elif retning == 'W':
        x -= 1
    else:
        y -= 1
    return (x == orm_x[sisteX]) and (y == orm_y[sisteY])
```



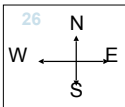
	0	1	2	3	4	5	6	7	8	9
0	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1
2	1	1	1	0	1	1	1	1	1	1
3	1	1	1	0	1	1	1	1	1	1
4	1	1	0	0	1	1	1	1	1	1
5	1	1	1	1	1	1	1	1	1	1
6	1	1	1	1	1	1	1	1	1	1
7	1	1	1	1	1	1	1	1	1	1
8	1	1	1	1	1	1	1	1	1	1
9	1	1	1	1	1	1	1	1	1	1

Spillbrettet en kvadratisk tabell av en vilkårlig størrelse, der 1 indikerer en tom rute.
Selve ormen representeres av to tabeller for hver akse (x og y) som tilsvarer lengden på ormen.
På figuren til venstre er ormen tegnet inn med tegnet '0' og er av lengde 5.
Tabellen for denne ormen er:

orm_x=[3,3,3,2,1]
orm_y=[2,3,4,4,4]

Hodet på ormen er markert med mørkest farge og vil da være punktet (3,2). Halen vil være punktet (1,4).

f) (10%) Skriv funksjonen ***flytt_orm*** som tar inn tabellene for ormen (orm_x og orm_y) samt retningen ormen skal bevege seg (N, E, S eller W). Funksjonen skal flytte posisjonene i de to tabellene slik at ormen beveger seg i riktig retning. Hvis vi tar utgangspunkt i verdiene gitt i tabellene orm_x og orm_y ovenfor, så vil for eksempel en forflytning i retning N (Nord) gi tabellene: orm_x = [3,3,3,3,2] og orm_y = [1,2,3,4,4]



	0	1	2	3	4	5	6	7	8	9
0	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1
2	1	1	1	0	1	1	1	1	1	1
3	1	1	1	0	1	1	1	1	1	1
4	1	1	0	0	1	1	1	1	1	1
5	1	1	1	1	1	1	1	1	1	1
6	1	1	1	1	1	1	1	1	1	1
7	1	1	1	1	1	1	1	1	1	1
8	1	1	1	1	1	1	1	1	1	1
9	1	1	1	1	1	1	1	1	1	1

Spillbrettet en kvadratisk tabell av en vilkårlig størrelse, der 1 indikerer en tom rute.
Selve ormen representeres av to tabeller for hver akse (x og y) som tilsvarer lengden på ormen.
På figuren til venstre er ormen tegnet inn med tegnet '0' og er av lengde 5.
Tabellen for denne ormen er:

orm_x=[3,3,3,2,1]
orm_y=[2,3,4,4,4]

Hodet på ormen er markert med mørkest farge og vil da være punktet (3,2). Halen vil være punktet (1,4).

f) (10%) Skriv funksjonen ***flytt_orm*** som tar inn tabellene for ormen (orm_x og orm_y) samt retningen ormen skal bevege seg (N, E, S eller W). Funksjonen skal flytte posisjonene i de to tabellene slik at ormen beveger seg i riktig retning. Hvis vi tar utgangspunkt i verdiene gitt i tabellene orm_x og orm_y ovenfor, så vil for eksempel en forflytning i retning N (Nord) gi tabellene: orm_x = [3,3,3,3,2] og orm_y = [1,2,3,4,4]

```
def flytt_orm(orm_x, orm_y,retning):
    dx,dy = 0,0
    lengde = len(orm_x)
    if retning == 'E':
        dx = 1
    elif retning == 'S':
        dy = 1
    elif retning == 'W':
        dx = -1
    else:
        dy = -1
    for i in range(lengde-1,0,-1):
        orm_x[i] = orm_x[i-1]
        orm_y[i] = orm_y[i-1]
    orm_x[0] += dx
    orm_y[0] += dy
```

g) (10%) Bruk metodene angitt ovenfor for å skrive kode for å gjøre følgende:
1.Opprett en heltalls variabel *score* som tar vare på poeng i spillet og sett den lik 0.
2.Opprett et *spillbrett* (tabell) med størrelsen 10x10 og fyll den med 1'er
3.Opprett tabellene orm_x og orm_y for å representere ormen som skal ha følgende verdier: Hode er punktet (3,4), midten er punktet (2,4), og halen er punktet (1,4).
4.Legg ut 5 epler på spillbrettet.
5.Sjekk posisjonen om posisjonen til hode på ormen på spillbrettet har mat (tallet 5), og øk variabelen *score* med 10 hvis dette er tilfellet.
6.Sjekk om ormen krasjer i seg selv hvis den beveger seg nordover (N) og minsk variabelen *score* med 5 hvis dette er tilfelle.
7.Flytt ormen en posisjon i retning nordover (N)
8.Tegn ormen inn i spillbrettet
9.Skriv ut spillbrettet til skjermen ved hjelp av *print der 0 skal representeres med tegnet ".", eple (5) med "*" og ormen med tegnet "O".*

Rekursjon

Rekursjon

- Rekursiv funksjon kalles det når en funksjon kaller seg selv.
- Rekursive funksjoner brukes ofte i algoritmer for å lage elegante løsninger på problemer.
- For at en rekursiv funksjon ikke skal fortsette i det uendelige, er det viktig at argumentet til funksjonen endres slik at den en gang får en stopp verdi.

rekursjon.py - /Users/terjery/Library/Mobile Documents/com~apple~CloudDo...

```
def count_down(antall):
    if (antall>0):
        print("Teller ned:",antall)
        count_down(antall – 1)


count_down(5)
```

Ln: 1 Col: 0

rekursjon.py

NB

- Noe som kan løses rekursivt kan også **alltid** løses iterativt

rekusjon.py - /Users/terjery/Library/Mobile Documents/com~apple~CloudDocs/ITGK...

```
def count_down(antall):
    if antall>0:
        print('Teller ned:',antall)
        count_down(antall-1)

count_down(5)
```

Ln: 7 Col: 0

tell_ned.py - /Users/terjery/Library/Mobile Documents/com~apple~CloudD...

```
def count_down(antall):
    for i in range(antall,0,-1):
        print(i)

count_down(5)
```

Ln: 6 Col: 0

Rekursjon til å kalkulere fakultet

- Fakultet til et tall beregnes på følgende måte:
  - Hvis n=0 så er n!=1
  - Hvis n>0 så er n!=1 * 2 * 3 * … * n
- For å lage en algoritme for fakultet må:
  - Først finne delen som ikke er rekursiv (som ikke skal kalles på nytt):
    - Hvis n = 0 så er n!=1 (denne er ikke avhengig av tidligere ledd)
  - Resten er den rekursive delen som stegvis kan beskrives:
    - Hvis n>0 så er fakultet(n) = n x fakultet(n-1)
  - Dette er alt vi trenger for å lage en rekursiv funksjon for å beregne fakultet

fakultet.py

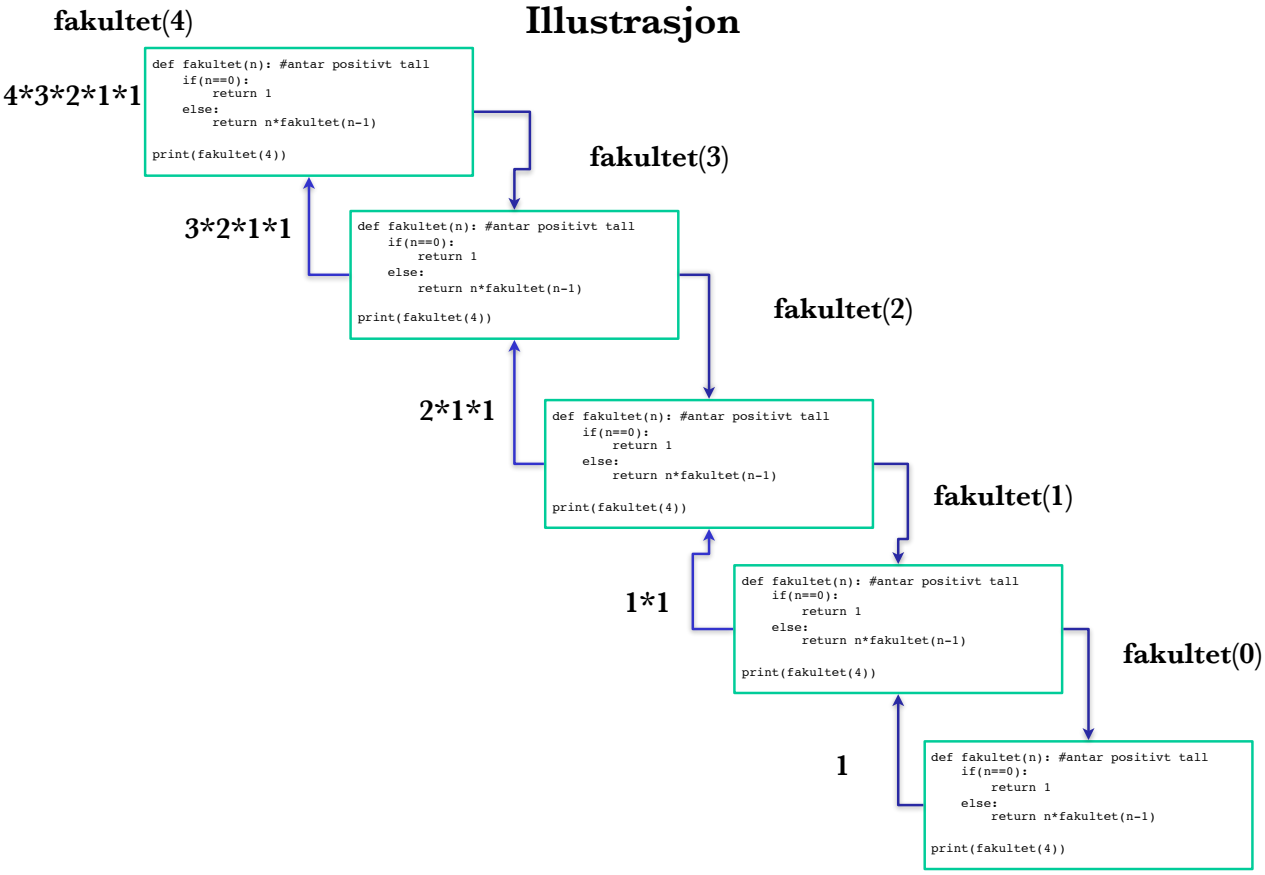
fakultet.py

fakultet.py - /Users/terjery/Library/Mobile Documents/com~apple~C...

```
def fakultet(n):
    if(n==0):
        return 1
    else:
        return n * fakultet(n-1)

print(fakultet(4))
```

Ln: 1 Col: 0

Illustrasjon



fakultet.py

Rekursivt

fakultet.py - /Users/terjery/Library/Mobile Documents/com~apple~C...

```
def fakultet(n):
    if(n==0):
        return 1
    else:
        return n * fakultet(n-1)

print(fakultet(4))
```

Ln: 1 Col: 0

Iterativt

fakultet.py - /Users/terjery/Library/Mobile Documents/com~apple~CloudDo...

```
def fakultet(n):
    prod = 1
    for i in range(1,n+1):
        prod *=i
    return prod

print(fakultet(4))
```

Ln: 8 Col: 0

Binærsøk (Binary Search Algorithm)

- Theory Book IT Grunnkurs
- Kapittel: Algorithm, 5.5

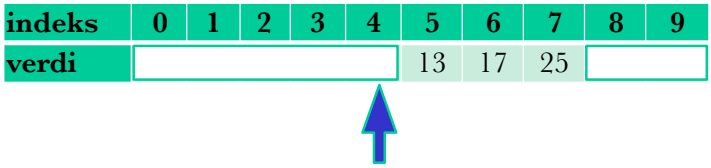
Binærsøkealgoritmen

- Binærsøkealgoritmen fungere på samme måte som vi kan spille gjettespillet fra uke 39.
  - Gjetter på den midterste verdien, og fjerner dermed halvparten av mulighetene for hver gjetting.

gjettespill.py

Illustrasjon av binærsøk

Finnes 17 i listen?



Sjekk det midterste elementet. (0 + 9)//2 -> 4, altså a(4)

17 > a(4)

Sjekk da det midterste elementet mellom a(5) og a(9) (5 + 9)//2 -> 7

17 < a(7)

Sjekk da det midterste elementet mellom a(5) og a(7) (5 + 7)//2 -> 6

17 == a(6)

17 finnes i listen

Rekursiv algoritme for binærsøk

```
bin_search(liste, verdi, min, max)
  hvis max < min Returner ikke funnet verdi
  hvis ikke
    midtpunkt = midtpunkt(min,max)
    hvis verdi < liste[midtpunkt]
       returner bin_search(liste,verdi,min,midtpunkt-1)
    hvis verdi > liste[midtpunkt]
       returner bin_search(liste,verdi,midtpunkt+1,max)
    hvis ikke
       returner midtpunkt
```

binary_search.py

binary_search.py

binary_search.py - /Users/terjery/Library/Mobile Documents/com~apple~CloudDocs/ITGK 2016/Python/Keynote/Uke 45/Python/binary_search.py (3.5.1)

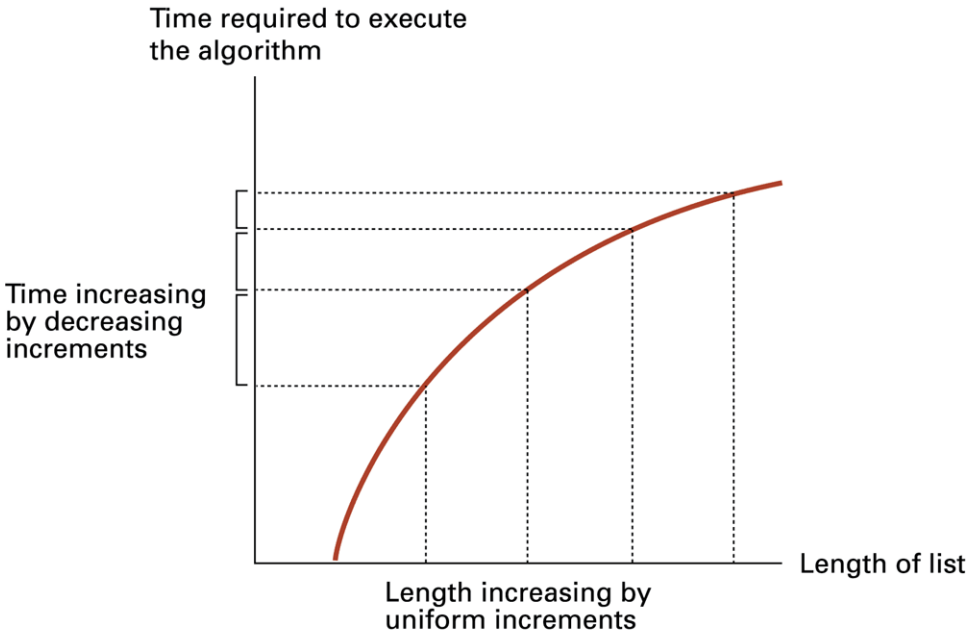
```
# Name: binary_search
# Input: a list and a search value
# Ouput: True if searchValue is found, else False
# Description: Search through a sorted list using the binary search algorithm

def binary_search(liste, verdi, imin, imax):
    if(imax < imin):
        return False
    else:
        imid = (imin+imax)//2 # Midtpunktet
        if (verdi<liste[imid]):
            return binary_search(liste,verdi,imin,imid-1)
        elif (verdi>liste[imid]):
            return binary_search(liste,verdi,imid+1,imax)
        else:
            return imid

A = [1,2,3,9,11,13,17,25,57,90]
result = binary_search(A,57,0,len(A)-1)
print(result)
```

Ln: 1 Col: 0

Karakteristikker av binærsøk

- Hva er best-case?
  - Treffer på midten første gang
  - Tidsbruk: $\Omega(1)$
- Hva er worst-case?
  - Halverer lista helt til det er ett element igjen
  - Tidsbruk: $O(\log n)$
- Kan sekvensielt søk være raskere enn binærsøk?
  - Ja hvis sekvensielt søk treffer tidlig i lista
- Begrensninger på binærsøk:
  - Lista må være sortert (pre-prosessert)

Graf over worst-case analyse av binært søk



GCD - Greatest Common Divisor

- Euclids algoritme bruker
  - divisjon og observasjonen at GCD mellom 2 tall er også en faktor i forskjellen mellom tallene
  - gcd(48,18)
    - divider 48 med 18. Dette gir 2 med en rest på 12
    - divider 18 med 12. Dette gir 1 med rest 6.
    - divider 12 med 6. Dette gir en rest på 0, hvilket betyr at 6 er gcd.
- Euclids algoritme for å finne største fellesnevner for 2 tall skrevet litt mer formelt:
  - gcd(a,0) = a
  - gcd(a,b) = gcd(b, a mod b)

Som Python-program

```
# gcd funksjonen returnerer den største fellesnevneren til to tall

def gcd(x, y):
    if x % y == 0:
        return y
    else:
        return gcd(y, x % y)
```

Rekursivt

gcd.py - /Users/terjery/Library/Mobile Documents/com~apple~CloudDocs/ITGK 2016/Python/1Forelesni...

```
def gcd(x, y):
    if x > y:
        storst = x
        minst = y
    else:
        storst = y
        minst = x
    teller = minst
    gcd = 1
    while True:
        if storst % teller == 0 and minst % teller == 0:
            break
        else:
            teller -=1
    return teller
```

Ln: 27 Col: 0

Iterativt