Rekursjon: Funksjoner som kaller på seg selv

Det enkleste er å begynne med et eksempel: En funksjon som skriver ut en stadig økende liste av påfølgende tall.

```
biggest_number = 10

def print_number(number) :
    if (number <= biggest_number) :
        print(number,end=" ")    # Vi skriver ut
        print_number(number+1)  # Vi kaller på oss selv med tallet, bare økt med 1
    else :
        return

print_number(1)
```

Vi kan se på et annet mer matnyttig eksempel: Regne ut n! som dere kjenner fra matematikken.

Dere vet også at $n! = n * (n-1) * (n-2) * .... * 2 * 1$

Dette kan enkelt løses med en for loop i python

```
n = input("Gi meg et positivt heltall")
produkt = 1
for i in range(int(n)) :
    produkt = produkt*i

print ("Svaret er ", produkt)
```

MEN: Dette kan også løses på en annen måte, vha. rekursjon. La oss benytte oss av at vi kan omskrive fakultet som

1. $n! = n * (n-1)!$
2. $(n-1)! = (n-1) * (n-2)!$
3. $(n-2)! = (n-2) * (n-3)!$ osv.

Vi ser at vi har en rekursiv regel her, der beregning i øverste steg avhenger av samme beregningen i neste steg, som igjen avhenger av samme beregning i neste steg osv... Helt til man kommer ned til tallet 1 som er slutt-betingelsen. Hvordan kan vi implementere dette vha. rekursjon?

```
# Rekursjon starter med største tallet i produktet
def factorial(n) :
    if (n > 0) :
        return n*factorial(n-1)
    else :
        return 1

factorial(10)
```

Vi ser at vi MÅ ha en grein i kontrollflyten som stopper rekursjonen.

- Kalles basis-operasjonen
  - Grunnen til dette er at det er den enkleste varianten som kan bli spurt etter i kallet til den rekursive funksjonen
  - Kalles også stopp-betingelsen eller "stopping condition"

Den andre greina tar seg av **sub-problemet**, der gjerne det rekursive kallet skjer.

Oppgaver til dere:

1. Lag en rekursiv funksjon for å regne ut 2^n
2. Lag en rekursiv funksjon for å regne ut summen av alle tallene fra 1 til n.