Oppgaver uke 5:

Merknad:
Mange av disse algoritmneoppgavene kan vi løse relativt lett ved hjelp av innebygde funksjoner i Python som *len*, *count*, *max*, *min*, osv. Vi vil at dere skal forstå hvordan man kan implementere disse funksjonene, så prøv å ikke bruke Python sine innebygde funksjoner, for eiga øvings skyld.

Funksjoner:

1. Lag en funksjon som tar en liste med tall og et tall som argument og returnerer en liste med alle elementene i den medsendte listen som er større enn det medsendte tallet

Løsningsforslag:

```
def stoerreEnn(talliste, tall):
    nyListe = []
    for etTall in talliste:
        if etTall > tall:
            nyListe.append(etTall)
    return nyListe
```

Vi må definere ei ny tom liste som vi kan legge elementene i etter hvert som vi finner dem. Så går vi igjennom alle tallene i lista med ei for-løkke. For hvert tall i lista sjekker vi om det er større enn tallet brukeren skrev inn. Hvis det er større, legger vi det i lista vår. Til slutt *returnerer* vi den lista.

2. a) Lag en funksjon som tar en liste med tall som argument og returnerer summen av tallene

   b) Lag en funksjon som tar en liste med tall som argument og returnerer gjennomsnittet av tallene
   def

Løsningsforslag:

```
#2a
def summer(liste):
    summen = 0
    for tall in liste:
        summen += tall
    return summen
```

Her tar vi i bruk ei for-løkke for gå gjennom hvert element i lista. Variabelnavnet *tall* er det navnet vi gir til hvert element i lista når vi går igjennom det. Denne variabelen vil altså overskrives for hver gjennomgang i løkka. Hvis vi har ei liste [1,5,3], vil verdien til *tall* være 1 første gang, 5 neste gang, og så 3 den siste gangen.
Vi trenger å definere en variabel før løkka for å kunne holde på summen etter hvert som den øker. Husk at *summen += tall* er det samme som *summen = summen + tall*.

```
#2b
def gjennomsnitt(liste):
    return summer(liste)/len(liste)
```

Siden vi definerte *summer* i 2a, kan vi bruke den her. Gjennomsnittet av ei liste med tall av lengde *n*, er **summen av alle elementene** i lista delt på **lengden av lista**.

3. a) Lag en funksjon som tar imot et tall som argument og returnerer kvadratet av tallet
b) Lag en funksjon som tar imot tre tall som argument og returnerer summen av de medsendte tallenes kvadrater

(optional)
c) Lag en funksjon som tar imot en liste, og returnerer en ny liste hvor hvert element er opphøyd med 2.

Løsningsforslag:

```
#3a alternativ 1
def kvadrat(tall):
    return tall * tall

#3a alternativ 2
def kvadrat(tall):
    return tall ** 2
```

Her tar vi inn et tall, og så returnerer vi tallet ganget med seg selv. Dette kan vi gjøre på de to måtene her. Vi kunne også mellomlagret kvadratet i en egen variabel, som kvadratTall = tall * tall, og så returnert dette tallet, men det er ikke nødvendig. Vi har ikke bruk for å mellomlagre verdien når vi uansett bare skal returnere den.

```
#3b alternativ 1

def kvadratSum(tall1,tall2,tall3):
    return kvadrat(tall1) + kvadrat(tall2) + kvadrat(tall3)

#3b alternativ 2
def kvadratSum(tall1,tall2,tall3):
    return tall1 ** 2 + tall2 ** 2 + tall3 ** 2
```

Disse to løsningene er like. Jeg tenkte at siden vi blir bedt om å lage en funksjon som regner ut kvadratet av tall, kan det være greit å vise at vi kan bruke den i den andre funksjonen, men i dette tilfellet sparer vi ikke mye på det. I mange tilfeller vil vi kunne spare mye på å kalle på andre funksjoner inne i andre funksjoner for å gjøre deler av jobben. Det kan også bli lettere å lese noen ganger.

#3c obs! Denne tok vi ikke i timen, jeg tror den ble lagt til i oppgavesettet senere.

```
def kvadratListe(talliste):
    nyListe = []
    for tall in talliste:
        nyListe.append(kvadrat(tall))
    return nyListe
```

Her tar vi altså inn ei liste. Vi lager ei ny, tom liste som vi kan putte de nye talla i. Så går vi gjennom hvert tall i lista, og legger til kvadratet av det tallet. Her kunne vi også selvsagt ha skrevet:

```
        nyListe.append(tall ** 2)

eller

        nyListe.append(tall * tall)
```

4. *Litt vanskelig:* Lag en funksjon som tar imot en liste og returnerer det nest største tallet

Løsningsforslag:

Etter å ha snakket med noen av dere er det to løsninger jeg vil anbefale. Den ene er litt lenger, men interessant. Den andre er litt kortere. Det vi gjør er at vi omformulerer spørsmålet. Det nest største tallet i ei liste er det største tallet etter at vi har fjernet det største tallet. Vi antar at lista vi tar inn bare har ulike tall.

Løsningsmulighet 1

```
def nestStoerst(liste):
    nyListe = []
    for tall in liste:
        nyListe.append(tall)
    stoerst = nyListe[0]
    for tall in nyListe:
        if tall > stoerst:
            stoerst = tall
    nyListe.remove(stoerst)
    nestStoerst = nyListe[0]
    for tall in nyListe:
        if tall > nestStoerst:
            nestStoerst = tall
    return nestStoerst
```

Først kopierer vi lista til ei ny liste, slik at når vi sletter det største tallet, så endrer vi ikke noe i den opprinnelige lista. Så sier vi at *stoerst* skal være det første elementet i lista. Det er fordi at det vil aldri gi et problem, fordi hvis det er det største, så er det allerede riktig, og hvis ikke, så vil det være et tall i lista som er større enn det. Det er vanskelig å finne en annen verdi vi kan sette *stoerst* til som alltid vil gå.

Etter dette fjerner vi det største tallet fra lista. Merk at vi har antatt at det ikke finnes to like tall i lista. Hvis ikke må vi bestemme oss for om vi vil at "nest størst" skal kunne være det samme som det største, eller om verdien skal være lik.

Så finner vi det største tallet etter slettinga. Nå kaller vi variabelen *nestStoerst*. Vi kunne fortsatt å bruke *stoerst*, men det er viktig at vi setter verdien til det første elementet i lista igjen, for ingen av tallene i den nye lista vil være større enn det største tallet i lista fra før slettinga.

Så returnerer vi *nestStoerst*.

Det er noen forenklinger vi kan gjøre. Vi ser at vi har skrevet relativ lik kode to ganger, vi har bare endret navnet på den ene variabelen. Her kan vi skrive en egen funksjon *stoerst*, som vi kan bruke:

```
def stoerst(liste):
    stoerst = liste[0]
    for tall in liste:
        if tall > stoerst:
            stoerst = tall
    return stoerst
```

I tillegg kan vi kopiere lista ved å bruke *slicing*. Det står om det i boka, og dere som tar IN1140 parallelt med IN1000 har kanskje vært borti det. Se side 328 i læreboka.Vi kan skrive:

```
    nyListe = liste[:]
```

Det er altså det samme som den for-løkka vi brukte til å kopiere med.

Nå kan vi skrive om funksjonen vår:

```
def nestStoerst2(liste):
    nyListe = liste[:]
    nyListe.remove(stoerst(nyListe))
    return stoerst(nyListe)
```

Om dere ikke vil bruke slicing er det helt fint, men for å unngå å repetere kode anbefaler jeg å skrive en egen stoerst()-funksjon.

Vi kan også skrive en versjon på få linjer hvis vi "jukser" litt. Hvis vi også vil tillate lister med mange like elementer, men vi vil at det nest største elementet skal være det som har nest størst verdi, så selv om det er to av det største tallet, skal vi finne det tallet som er nærmest det største, men med lavere verdi.

```
def nestStoerst3(liste):
    nyListe = list(set(liste))
    nyListe.sort()
    return nyListe[len(nyListe) - 2]
```

Det vi gjør her er å først konvertere lista som vi tar inn til en *mengde*. Husk at i mengder kan vi ikke ha like elementer, så vi får bare ett av hvert tall. Så konverterer vi tilbake til liste. Merk hvordan dette er nøsta inni hverandre.

Etter det sorterer vi ved å bruke den innebygde metoden .sort(). Den sorterer i rekkefølge fra lavest til høyest.

Da vet vi at det nest siste elementet må være det nest største, så vi kan returnere det som ligger på indeks [len(nyListe) - 2], altså lengden av lista minus 2.

Merk at funksjonen *set* returnerer en ny mengde, så vi trenger ikke å uroe oss for at vi endrer noe i den lista vi sender inn.

Løsningsmulighet 2

En annen mulighet er at i tillegg til å ha en variabel som holder på *stoerst* når vi sjekker hvilket tall som er stoerst, kan vi ha en variabel som holder på *nestStoerst*.

```
def nestStoerst(liste):
    stoerst = liste[0]
    nestStoerst = liste[0]
    for tall in liste:
        if tall > stoerst:
            nestStoerst = stoerst
            stoerst = tall
    return nestStoerst
```

Hver gang vi finner et tall som er høyere enn *stoerst*, sier vi at *nestStoerst* skal ha verdien til *stoerst*, og så oppdaterer vi verdien til *stoerst.*

5. *Litt vanskelig:* Lag en funksjon som tar imot to lister som argument og returnerer om listene er like eller ikke (om tallene i listene er de samme og i samme rekkefølge)

Løsningsforslag:

Her ber oppgaven oss om å returnere *om listene er like eller ikke*, det kan vi tolke som en boolsk verdi, altså True eller False.

```
def like(liste1,liste2):
    if len(liste1) != len(liste2):
        return False
    else:
        for i in range(len(liste1)):
            if liste1[i] != liste2[i]:
                return False
    return True
```

Først sjekker vi om lengden av listene faktisk er like. Hvis de ikke er det kan vi returnere *False*, hvis ikke kan vi gå videre.
Så bruker vi *for i in range()* på lengden av lista. range(len(liste1)) vil gi oss alle indeksene til listene. Vi vet at listene er like lange, så vi kan bruke dette til å sammenligne hvert element på samme indeks i begge listene samtidig. Vi trenger bare ett tilfelle der de ikke er like for at listene ikke skal være like, så vi kan returnere *False*.
Hvis vi ikke kommer til noen av *return False*-kallene, kan vi trygt returnere *True*. Da vet vi at listene er like.