

Løkker



Løkker er helt essensielle i koding.

Løkker



Løkker er helt essensielle i koding.
Idéen bak er å repetere en viss kodesnutt om og om igjen.

Løkker - while-løkken



Strengt tatt er alle løkker en while løkke.

Løkker - while-løkken



While-løkker kjører kun hvis en boolsk verdi er True. Det betyr at vi kan sjekke en condition i løkka.

```
while True:
    print("to infinity!")
print("and beyond?")
```

Løkker - while-løkken



Minner om en for-løkke?

```
i = 0
while i < 10:
    print(i)
    i += 1
```

Løkker - while-løkken



Veldig bra å bruke dersom man ikke vet hvor mange ganger løkka skal kjøre, men *vet det når man ser det*.

```
# dette er en metode som sikrer at det tomme svaret
# ikke blir tatt med
answer = input("Skriv inn ditt svar: ")
answer_lst = []
while answer != "": # tomt svar, trykket enter med en gang
    answer_lst.append(answer)
    answer = input("Skriv inn ditt svar: ")
```

Løkker - for-løkken



for-løkker brukes når man har en bestemt *iterable* man ønsker å iterere gjennom.

Løkker - for-løkken



for-løkker brukes når man har en bestemt *iterable* man ønsker å iterere gjennom. Dette gjelder spesielt de vi allerede har gått gjennom, i form av *lister* og *tupler*.

Løkker - for-løkken



for-løkker brukes når man har en bestemt *iterable* man ønsker å iterere gjennom. Dette gjelder spesielt de vi allerede har gått gjennom, i form av *lister* og *tpler*.

Disse kan defineres *inline*

```
for word in ("hei", "hei", "alle", "sammen"):
    print(word)
for word in ["hei", "hei", "alle", "sammen"]:
    print(word)
```

Løkker - for-løkken



for-løkker brukes når man har en bestemt *iterable* man ønsker å iterere gjennom. Dette gjelder spesielt de vi allerede har gått gjennom, i form av *lister* og *tpler*.

Eller på forhånd. Da kan de gjenbrukes senere, eller redefineres før.

```
tpl = ("hei", "hei", "alle", "sammen")
lst = ["hei", "hei", "alle", "sammen"]
for word in tpl:
    print(word)
for word in lst:
    print(word)
```

Løkker - for-løkken



Det er hovedsakelig to måter å *iterere* gjennom en liste.

Løkker - for-løkken



Det er hovedsakelig to måter å *iterere* gjennom en liste.
Element-vis. Man henter ut hvert *element* i listen.

```
lst = [1, 2, 3, "hei", (1, 2, 3)]  
for element in lst:  
    print(element)
```

Løkker - for-løkken



Det er hovedsakelig to måter å *iterere* gjennom en liste.
Indeks-vis. Man henter ut *plasseringen* til hvert element i listen.

```
lst = [1, 2, 3, "hei", (1, 2, 3)]  
for i in range(len(lst)):  
    print(lst[i])
```

Løkker - for-løkken



Det er hovedsakelig to måter å *iterere* gjennom en liste. Indeks-vis. Man henter ut *plasseringen* til hvert element i listen. Viktig: hvis vi itererer element-vis gjennom en liste, vet vi egentlig ingenting om plasseringen, og hvis vi trenger indeksen, er det best å gå gjennom indeks-vis.

```
lst = [1, 2, 3, "hei", (1, 2, 3)]  
for i in range(len(lst)):  
    print(lst[i])
```

Løkker - for-løkken



Det er hovedsakelig to måter å *iterere* gjennom en liste.
Kan bruke *enumerate* for å få ut begge med en gang.

```
lst = [1, 2, 3, "hei", (1, 2, 3)]  
for i, element in enumerate(lst):  
    # lst[i] == element -> True  
    print(i, lst[i], element)
```

Løkker - for-løkken



Det er hovedsakelig to måter å *iterere* gjennom en liste. Kan bruke *enumerate* for å få ut begge med en gang. Sjeldent dette ikke kan løses med bare indekser, og tilordning.

```
lst = [1, 2, 3, "hei", (1, 2, 3)]

for i, element in enumerate(lst):
    # lst[i] == element -> True
    print(i, lst[i], element)

for i in range(len(lst)):
    element = lst[i]
    # lst[i] == element -> True
    print(i, lst[i], element)
```


Løkker - for-løkken



Løkker kan enkelt skrives inne i hverandre.

```
def antall_like(lst1, lst2):  
    res = 0  
    for x in lst1:  
        for y in lst2:  
            if x == y:  
                res += 1  
  
    return res
```

Løkker - for-løkken



Løkker kan enkelt skrives inne i hverandre. Går gjennom hvert element i *lst1*, sjekker hvert element i *lst2* og ser om jeg finner det.

```
def antall_like(lst1, lst2):  
    res = 0  
    for x in lst1:  
        for y in lst2:  
            if x == y:  
                res += 1  
  
    return res
```

Løkker - for-løkken



Løkker kan enkelt skrives inne i hverandre. Går gjennom hvert element i *lst1*, sjekker hvert element i *lst2* og ser om jeg finner det. Her trenger vi ikke gå gjennom med indekser (range), siden vi trenger ikke vite noe om posisjon.

```
def antall_like(lst1, lst2):  
    res = 0  
    for x in lst1:  
        for y in lst2:  
            if x == y:  
                res += 1  
  
    return res
```

Løkker - for-løkken



Det kan se vanskelig ut å analysere når det er flere løkker.

```
def antall_like(lst1, lst2):  
    res = 0  
    for x in lst1:  
        for y in lst2:  
            if x == y:  
                res += 1  
  
    return res
```

Løkker - for-løkken



Det kan se vanskelig ut å analysere når det er flere løkker. Det er enklest å begynne å analysere *innenfra og ut*.

```
def antall_like(lst1, lst2):  
    res = 0  
    for x in lst1:  
        for y in lst2:  
            if x == y:  
                res += 1  
  
    return res
```

Løkker - for-løkken



Det kan se vanskelig ut å analysere når det er flere løkker. Det er enklest å begynne å analysere *innenfra og ut*. Dette gjelder også når vi skal designe kode som bruker kode inni kode.

```
def antall_like(lst1, lst2):  
    res = 0  
    for x in lst1:  
        for y in lst2:  
            if x == y:  
                res += 1  
  
    return res
```

Løkker - for-løkken



Vi antar at x og y er bestemte verdier (f.eks. $x = 1$, $y = 2$). Hva er det vi vil gjøre med denne bestemte verdien?

```
def antall_like(lst1, lst2):  
    res = 0  
    for x in lst1:  
        for y in lst2:  
            if x == y:  
                res += 1  
  
    return res
```

Løkker - for-løkken



Vi antar at x og y er bestemte verdier (f.eks. $x = 1$, $y = 2$). Hva er det vi vil gjøre med denne bestemte verdien?

Hvis disse er like, øker vi *telleren* med 1.

```
def antall_like(lst1, lst2):  
    res = 0  
    for x in lst1:  
        for y in lst2:  
            if x == y:  
                res += 1  
  
    return res
```


Løkker - for-løkken

Vi antar at x og y er bestemte verdier (f.eks. $x = 1$, $y = 2$). Hva er det vi vil gjøre med denne bestemte verdien?

Hvis disse er like, øker vi *telleren* med 1.

Nå er den *innerste* delen av koden abstrahert bort. Hva er neste steg?

```
def antall_like(lst1, lst2):  
    res = 0  
    for x in lst1:  
        for y in lst2:  
            if x == y:  
                res += 1  
  
    return res
```

Løkker - for-løkken



Vi antar at x er en bestemt verdi (f.eks. $x = 1$).

Vi vet hva som skjer for en bestemt y , så det kan generaliseres. Vi vet hva x er, og vi vet hva vi skal gjøre hvis vi har en x og en y . Så vi går gjennom alle mulige y -er.

```
def antall_like(lst1, lst2):  
    res = 0  
    for x in lst1:  
        for y in lst2:  
            if x == y:  
                res += 1  
  
    return res
```

Løkker - for-løkken

Nå trenger vi ikke lenger å se på en bestemt verdi for x , siden vi vet hva som skjer med en (den sammenlignes med hver mulige y), kan den også generaliseres.

For hver mulige x , sjekk hver mulige y . Dersom $x == y$, så øker vi telleren med en.

```
def antall_like(lst1, lst2):  
    res = 0  
    for x in lst1:  
        for y in lst2:  
            if x == y:  
                res += 1  
  
    return res
```

Løkker - for-løkken

Strenger og tupler kan behandles ganske likt, siden begge er ikke-muterbare. F.eks., dersom vi vil endre noe, så må vi bygge disse strukturene på nytt.

Dette kan vi gjøre litt enklere senere med slicing.

```
def insert_string(original, index, new_string):  
    # antar at index er valid  
    resultat = "" # starter med tom streng  
    for i in range(len(original)):  
        if i == index:  
            resultat += new_string  
        # om vi plusser foran eller bak new_string  
        # bestemmer om vi inserter foran eller bak.  
        resultat += original[i]  
    return resultat
```