Repetisjonskurs H21

Løkker og Lister

Dagens plan

- Samlinger
- Litt om boolske uttrykk
- While-løkker
- For-løkker

Bruk menti.com for spørsmål!

Spørsmål vil bli gjennomgått i time 2!

Bruk kode:

Samlinger

Lister

Hva er en liste?

- Brukes til å holde på og behandle flere verdier
- Hver plass har sin egen verdi – *Indeks*
  - Starter på 0 og øker i takt med størrelsen på lista
- Lister er dynamiske, vi kan endre størrelse og innhold når som helst etter ønske
- Sortert

- Opprette lister:

```
tom_liste = []                              # Oppretter en tom liste
liste_med_tall = [1,2,3]                    # Oppretter en liste med tall
liste_med_flere_typer = ["fire", 5, "seks"] # Oppretter en liste med flere typer
```

- Hente ut verdier på indeks

```
liste_med_flere_typer = ["fire", 5, "seks"]
print(liste_med_flere_typer[1]) >> 5
```

- Og utføre *operasjoner*

```
ansatte = []
deltakere.append("Micheal")
deltakere.append("Dwight")
deltakere.append("Jim")

print(ansatte) >> ["Micheal", "Dwight", "Jim"]
```

Eksempel: Handleliste

- Vi legger til varer etter hvert som vi kommer på hva vi trenger

```
handleliste = []              # Tom liste
handleliste.append("Melk")
handleliste.append("Egg")     # Legger til 3 ting
handleliste.append("Brød")
```

- Vi kan legge til på bestemte steder i listen dersom vi ønsker det

```
handleliste.insert(0, "Juice")  # Legger til på indeks 0
```

- Vi tar de av listen etter hvert som vi kjøper dem

```
handleliste.pop()             # Fjerner siste
handleliste.pop(0)            # Fjerner første
```

Oppgave:

- Opprett en tom liste.
- Legg til 6 elementer av valgfri type
- Skriv ut to elementer ved hjelp av indeks
- Fjern første og siste element
- Skriv ut lengden på lista

```
Listeoperasjoner

liste.insert(indeks, elem)   # Legger til på indeks
liste.append(elem)           # Legger til på siste
liste.pop(indeks)            # Fjerner og returnerer et element
liste.remove(elem)           # Fjerner et element
len(liste)                   # Gir lengden på lista
```

Mengder

- Nesten det samme som liste, men:
  - Kan ikke inneholde duplikater
  - Er uordnet (uten rekkefølge)
- Opprettes slik:

```
tom_mengde = set()
ikke_tom_mengde = {"en", "en", "to", "tre"}

print(len(ikke_tom_mengde))
```

(Hva skrives ut her?)

Fra liste til mengde

```
liste = [1,1,1,2,3,4,4,5]
len(liste) >> 8

mengde = set(liste)
len(set) >> 5
```

Operasjoner på mengder

```
mengde = set()
mengde.add(elem)      # Legger til element
mengde.discard(elem) # Fjerner element
mengde.remove(elem)   # Fjerner element
mengde.clear()        # Fjerner alle elementer
```

Ordbøker

- Holder på nøkkel-verdi par som henger sammen
- Kan sammenlignes med for eksempel en telefonbok:

```
{"Geir":12345678, "Mikkel":98765432}
```

- Kan opprettes med eller uten verdier:

```
tom_ordbok = {}

telefonbok = {"Geir":12345678, "Mikkel":98765432}

norsk_engelsk = {"hei":"hello", "hade":"goodbye"}
```

- Legge inn verdier i en ordbok:

```
dyr = {}

dyr["kanin"] = "pattedyr"
dyr["hai"] = "fisk"
```

- Hente ut fra ordbok:
  - Bruker nøkkelen -> får verdien

```
print(dyr["kanin"]) >> "pattedyr"
type_for_hai = dyr["hai"] >> "fisk"
```

Liste vs. Mengde vs. Ordbok

- Liste
  - Har rekkefølge, indeksaksessering, tillater duplikater
- Mengde
  - Ingen rekkefølge, ingen indeksaksessering, tillater ikke duplikater
- Ordbok
  - Ingen rekkefølge, aksessering med nøkler, ingen duplikater *på nøkler*
  - Raskere enn liste

Ekstra - Nøstet liste

- Lister inni lister
- Kan ses på som et koordinatsystem eller en matrise
- Eksempel – Koffert:

```
toalettmappe = ["tannbørste", "hårbørste", "tannkrem"]
klaer = ["jakke", "bukse", "sokker", "undertøy"]
div = ["kamera", "ladere", "laptop", "bok"]

koffert = [toalettmappe, klaer, div]
```

- Oppgave: Hvilke elementer er dette?

```
koffert[0][2] >> "tannkrem"
koffert[1][1] >> ?
koffert[2][0] >> ?
koffert[0][0] >> ?
```

- Oppgave: Hvilken indeks har disse?

```
"ladere"     >> ?
"sokker"     >> ?
"hårbørste" >> ?
```

Kort om boolske uttrykk

- Noe som *evaluerer* til enten True (truthy) eller False (falsy)
- Noen eksempler

```
"a" == "a" >> True
1 != 2     >> True      # Ikke lik
1 == "1"   >> False
1 < 1      >> False     # Ekte mindre
1 <= 1     >> True      # Mindre eller lik
```

Not

- Inverterer det boolske uttrykket

```
not(False) >> True
not(True)  >> False

not 5 == 5 >> ?
not 5 > 1  >> ?
not 5 > 20 >> ?
not 5 == 1 >> ?
```

(Hva blir resultatet på de siste?)

Vi kan sette sammen boolske uttrykk

AND

- Begge er True

```
False and False >> False
False and True  >> False
True and True   >> True

(5 == 1) and (5 > 20) >> ?
(5 == 1) and (5 > 1)  >> ?
(5 == 5) and (5 > 1)  >> ?
```

OR

- Den ene eller begge er True

```
False or False >> False
False or True  >> True
True or True   >> True

(5 == 1) or (5 > 20) >> ?
(5 == 1) or (5 > 1)  >> ?
(5 == 5) or (5 > 1)  >> ?
```

Løkker

While

- Repeterer så lenge et boolsk uttrykk er True

```
while <noe som er True/False>:
    <repeterende kode>
```

- Viktig å ha noe som avslutter løkken!
  - Noe som oppdaterer boolen
  - Unngår evig løkke

Det vi *ikke* skal gjøre:

```
while True:
    if <noe som er True>:
        break
    <repeterende kode>
```

Eksempel – Enkel kommandoløkke

```
brukerinput = input("Tast 'q' for å avslutte: ")
while brukerinput != "q":
    print("Fortsetter...")
    brukerinput = input("Tast 'q' for å avslutte: ")
```

Oppgaver - 10 minutter

Oppgave 1

- Skriv en kommandoløkke med 2 muligheter:
  - `q` skal avslutte programmet
  - `p` skal ta inn to tall fra brukeren og plusse dem sammen, og deretter skrive ut resultatet
  - **Alle andre inputs gjør at løkken kjører på nytt**

Oppgave 2

- Skriv en while-løkke som øker en tallvariabel med 1 for hver iterasjon, helt opp til hundre.
- Skriv ut alle tallene underveis

```
while <noe som er True/False>:
    <repeterende kode>
```

For-løkker

- Brukes vanligvis når vi vet nøyaktig hvor mange ganger noe skal kjøres.
- To typer!
- **For** med teller (med *range*)
- **For each** (for-løkke på samlinger)

For med teller

- Kjører *n* antall ganger med *range(n)*:

```
for <teller> in range(<antall ganger>):
    <repeterende kode>
```

- Eksempel – teller opptil et gitt tall:

```
def tell_til(tall):
    for i in range(tall):
        print(i)
```

- Merk! *Range* ikke inklusiv – teller ikke med siste. Forrige eksempel teller 0 – 9
  - Kan ta med +1 for å være inklusiv

```
def tell_til(tall):
    for i in range(tall+1):
        print(i)
```

- Range tar også flere argumenter:

```
for i in range(<start>, <slutt>, <antall hopp>)
```

For each (på samlinger)

- Itererer gjennom en hel samling:

```
for <variabelnavn> in <samling>:
    <repeterende kode>
```

- Eksempel – Skriv elementer i liste:

```
liste = [1,2,3,"banan",5,-1,3]

for element in liste:
    print(element)
```

Flere eksempler

To måter for liste

- Hvis vi trenger indeksen:

```
liste = [1,2,3,"banan",5,-1.3]

for indeks in range(len(liste)):
    print(liste[indeks])
```

- Hvis vi kun skal ha elementet:

```
liste = [1,2,3,"banan",5,-1.3]

for element in liste:
    print(element)
```

For ordbok og mengde

- Ordbok – to måter:

```
telefonbok = {"Geir":12345678, "Mikkel":98765432}
for key in telefonbok:
    print(key, telefonbok[key])

for key, value in telefonbok.items():
    print(key, value)
```

- Mengde:

```
mengde = {1,1,1,2,2,3,3,4}
for element in mengde:
    print(element)
```

Iterasjon i nøstet liste

- For å løpe gjennom en nøstet liste må vi bruke 2 løkker:

```
toalettmappe = ["tannbørste", "hårbørste", "tannkrem"]
klaer = ["jakke", "bukse", "sokker", "undertøy"]
div = ["kamera", "ladere", "laptop", "bok"]
koffert = [toalettmappe, klaer, div]

for liste in koffert:
    for ting in liste:
        print(ting)
```

- Vi kan også bruke indeksene:

```
for y in range(len(koffert)):
    for x in range(len(koffert[y])):
        print(koffert[y][x])
```

- Her er:
  - Y = indeks for de indre listene
  - X = indeks for tingene i de indre listene
- Legg merke til at det er omvendt fra et vanlig koordinatsystem!

Oppgave - Battleships

- Vi skal lage en enkel utgave av Battleships
- Ett brett, én spiller
- 5x5
  - `O` for et skjult rute
  - `-` for et skip
  - `X` for et treff

```
Velkommen til Battleships, spiller!
O O O O O
O O O O O
O O O O O
O O O O O
O O O O O
```

- Programmet kjører så lenge det fremdeles er skip på brettet (tips: kan være lurt med teller for antall skip)
  - Ta inn x og y som koordinater (obs! Omvendt i nøstede lister)
  - Sjekk om det er et skip der
    - Hvis ja – gi tilbakemelding om treff, oppdater brettet slik at treffet er synlig
    - Hvis nei – gi tilbakemelding om bom