GRUPPETIME
UKE 3

IN1000 GRUPPE 25 - 13.09.21

PLAN FOR GRUPPETIMEN

- Oblig 2 - hvordan gikk det?
- Objekter tilbyr tjenester
- Jobbe sammen på MetroRetro
- Lister
- Nøstede lister
- Mengder
- Ordbøker

LÆRDOMMER FRA OBLIG 2

- Kodeflyt
  - Når programmet kjøres - hvilken rekkefølge utføres kodelinjene i gitt x antagelse(r)?
  - Se på Trix. 02.14
- Kjører programmet?
  - Svaret kan være både ja og nei gitt forskjellige antagelser om hvordan programmet kjøres!
- Feilmeldinger
  - Se på understanding error messages
- Bruk prosedyrer for å forenkle kode!
- Unngå laaange kommentarer → prøv å ikke overskrid hjelpestreken i Atom på høyre side

LÆRINGSMÅL UKE 3

- Vite om begrepet objekter, samt at objekter tilbyr tjenester
- Kunne bruke samlinger for å holde på mange verdier
- Kjenne til forskjellene mellom lister, mengder og ordbøker, samt være i stand til å benytte egnet type samling for et gitt formål

OBJEKTER tilbyr tjenester

- Et objekt tilhører en klasse
  - en tekststreng er et **objekt** av klassen str
  - volvo kan være et objekt av klassen Bil - vi kan lage våre egne klasser
- Gjennom **metoder** tilbyr objekter tjenester
  - en metode ligner på en funksjon/prosedyre - men den er alltid knyttet til et objekt
  - kalle på en metode med dot-notasjon
    - streng.upper()

NOEN NYTTIGE METODER SOM STR TILBYR

```
hei = "heisann sveisANN!"
hei.upper() # STORE BOKSTAVER
hei.lower() # små bokstaver
hei.count("n") # Teller forekomster
hei.find("sveisANN") # Finds first index of occurance of value
hei.replace("heisann", "hade") # Erstatter heisann med hade
hei.title() # Gjør om til overskrift -> Heisann Sveisann

bare = "    fjern   BARE mellomrom     foran og bak     "
bare.strip() # Fjerner blanklines foran og bak i strengen

frokost = "banan, yoghurt, pannekaker, bacon, cereal"
print(frokost.split(", ")) # Splitter en streng til en liste på x
```

Les om flere str metoder her

HVA SLAGS SAMLING?

Ordene i en setning	liste med stringer, fordi rekkefølgen er viktig og et ord kan forekomme to ganger i en setning
De latinske navnene på alle forskjellige spiselige soppene i Norge	mengde med stringer, fordi vi ikke vil ha duplikater
Navnene på alle klassekameratene dine, i alfabetisk rekkefølge	liste med stringer, fordi rekkefølgen er viktig og et navn kan vises to ganger
Hvilket måltid du skal spise for hver dag i uken	en ordbok med hverdager som nøkkel (string eller heltall) og måltid som verdi (string)
Telle hvor mange fugler du ser av hver art i hagen din	ordbok med fuglearter som nøkler og tall som verdi
Alle primtall under 1000	mengde med heltall, fordi rekkefølgen er ikke viktig og vi vil ikke ha duplikater

Fra Trix oppgave 03.11 - les mer om lister, mengder og ordbøker her

LISTER

- Lister har dynamisk størrelse
  - Dersom man legger til ett element blir listen en størrelse større, dersom man fjerner ett element blir listen en størrelse mindre.
- Man kan ha lister med forskjellige elementer:
  - tall_liste = [2, 4, 6, 7]
    navne_liste = [“Anne”, “Per”, “Lisa”]
    misc_liste = [“A”, 1, 9.2, “dette er bare ett element i lista”]
    tom_liste = []
- MetroRetro: https://metroretro.io/board/LB0J6GFYJZQ4

Oppgave 1: Hvor mange elementer?

```
liste = [0]                                    # 1

liste1 = [1, 2, 3]                             # 3

liste2 = [3, 3, 5, 7]                          # 4

liste3 = ["A", "BC", "D", "E", "F"]            # 5

liste4 = ["mange elementer i denne listen"] # 1

liste5 = []                                    # 0/tom
```

Vi kan bruke len(liste) for å finne antall elementer i en liste.

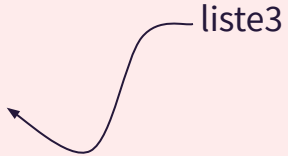
Oppgave 2: liste indeksering

```
# 2.1 Hva ligger på indeksene?
liste3[4] # "F"
liste1[0] # 1
liste4[1] # "mange elementer i denne listen"
liste3[5] # ingenting, er kun 5 elementer, på indeksene 0-4.

# 2.2 På hvilken indeks ligger
liste2[2] # ← 5 i liste2
liste3[1] # ← "BC" i liste3
liste1[0] # ← 1 i liste1
liste3[3] # ← "E" i liste3
```

```
liste = [0]
liste1 = [1, 2, 3]
liste2 = [3, 3, 5, 7]
liste3 = ["A", "BC", "D", "E",
"F"]
liste4 = ["mange elementer.."]
liste5 = []
```

0	“A”
1	“BC”
2	“D”
3	“E”
4	“F”



Oppgave 3: liste indeksering

```
navn = ["Ola", "Martin", "Selma"]
navn.append("Kari")
# 3.2 ["Ola", "Martin", "Selma", "Kari"]

print(navn[1])
navn.insert(0, "Lise")
navn.insert(0, "Kai")
# 3.4 ["Kai", "Lise", "Ola", "Martin", "Selma", "Kari"]
navn.remove("Martin") # alternativt navn.pop(3)

print("Alle navn: ", navn)

# Bonus:["Kai", "Lise", "Ola", "Selma", "Kari"]
```

0	“Kai”
1	“Lise”
2	“Ola”
3	“Martin”
4	“Selma”
5	“Kari”

Operasjoner på lister

```
len(liste) # Gir lengden på listen
liste.insert(index, element) # Sette inn element på angitt index
liste.append(element) # Setter inn nytt element på slutten av
listen
liste.remove(element) # Fjerne angitt element fra listen
liste.pop(index) # Fjerner (og returnerer) element på angitt indeks
liste.pop(-1) # Da slettes det siste elementet(!)
print("Liste: ", liste) # Skriver ut> Liste: 1, 2, 3
```

Les om flere liste metoder her

NØSTEDE LISTER

- Kan sees på som lister i lister

Koffert eksempel

- Vi skal ha med oss toalettsaker, klær og en liten bag med diverse

```
toalettsaker = ["tannborste", "haarborste", "tannkrem"]
klaer = ["jakke", "bukse", "sokker", "undertoy"]
div = ["kamera", "lommebok", "pass", "mobillader"]
```

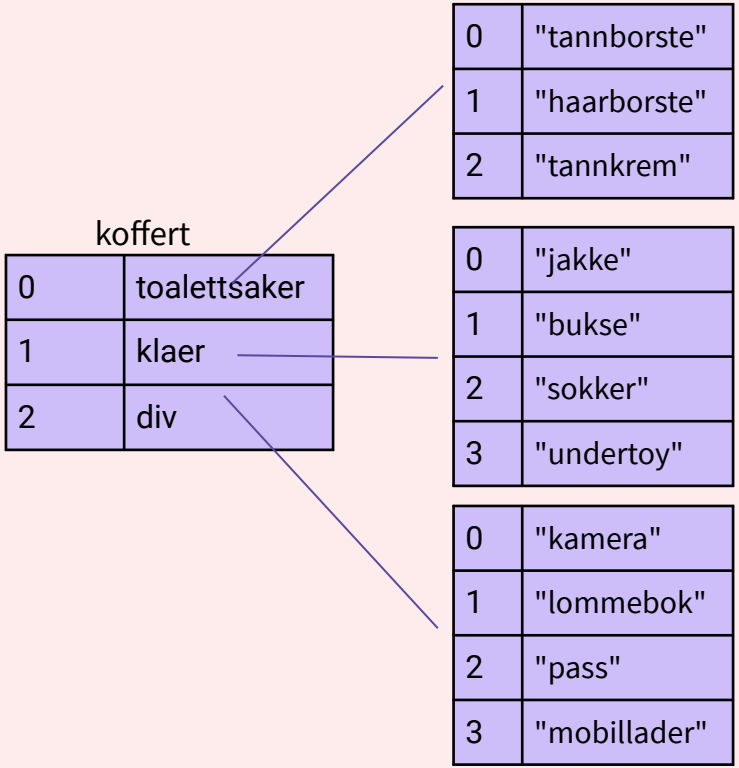
- For å samle alle småbaggene når vi skal på tur legger vi dem i en koffert:

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

Nøstede lister: koffert eksempel

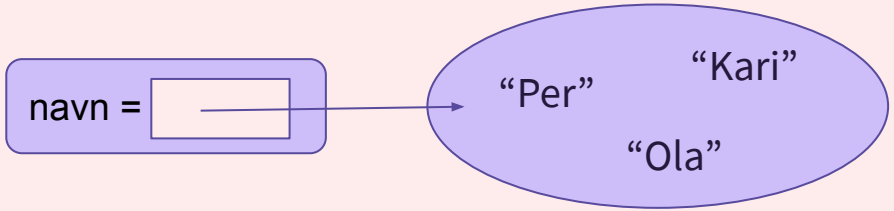
Hvordan få tak i ett element?
koffert[0][2] → tannkrem
koffert[2][1] → lommebok
koffert[1][1] → bukse
koffert[2][0] → kamera
koffert[0][0] → tannborste

Hvordan henter man frem?
“mobillader” → koffert[2][3]
“sokker” →
“haarborste” →



MENGDER

- Lages nesten likt som en liste. Syntaks:
  - mengde = {element, element2, element3, ….}
    navn = {“Kari”, “Per”, “Ola”}
- Mengder har ingen spesifikk rekkefølge på elementene, ingen indeksering.



- Tom mengde lages slik: mengde = set()
  - IKKE mengde = {} ← *fordi dette er en tom ordbok!*

MENGDER VS. LISTER

MENGDER/SET	LISTER
navn = {“Kari”, “Per”, “Ola”} navn2 = {“Per”, “Ola”, “Kari”} if navn == navn2 : …. evalueres til **sant** fordi begge inneholder de samme elementene, sett er uavhengig av rekkefølgen Kan ikke ha duplikater Ingen spesifikk rekkefølge Kan ikke indekseres	navn = [“Kari”, “Per”, “Ola”] navn2 = [“Per”, “Ola”, “Kari”] if navn == navn2 : …. evalueres til **usant** selv om begge inneholder de samme elementene, så er de i ulik rekkefølge og dermed ikke like lister Kan ha duplikater Spesifikk rekkefølge Kan indekseres

Operasjoner på mengder

```
mengde = set()
mengde_av_liste = set(liste)
mengde.add(element) # Legge til nytt element i mengden
mengde.remove(element) # Elementet må være i mengden for å fjernes
mengde.discard(element) # Fjerner element fra mengden uansett
mengde.clear() # Fjerner alle elementer i mengden, gjør mengden tom
len(mengde) # Finner antall elementer i mengden.
```

Les om flere set metoder her

ORDBØKER

- Er en beholder som holder på par av nøkler og verdier.
  - Hver nøkkel i ordboken er knyttet til én verdi.
- Syntaks:
  ordbok = {nøkkel : verdi, nøkkel2 : verdi2, …..}
  - kontakter = {“Kari” : 47543234, “Ola” : 76554532, “Martin” : 99543367}
- I stedet for å bruke indeksering som i lister, bruker man nøkkelen (i dette eksempelet navn) for å hente ut verdier.
  - print(“Kari sitt telefonnummer er “, kontakter[“Kari”])
- tom_ordbok = {}

Operasjoner på ordbøker

```
tom_ordbok = {}

# Legge til nytt element i ordboken ELLER endre på
eksisterende
ordbok[nokkel] = verdi
# Hvis "Per" finnes i ordboken vil verdien til Per endres
kontakter["Per"] = 45667990

# Fjerne nøkkelen og verdien
ordok.pop(nokkel)
kontakter.pop("Per")
```

Les om flere ordbok metoder her

Oppgave 5: ordbøker

```
# 5.1 definerer ordboken brukere
brukere = {"hanjo" : "Hanne Johansen", "karsi" : "Kari Sirisen", "olha" : "Ole
Hansen"}

# 5.2 endrer navnet til karsi
brukere["karsi”] = "Kari Marie Sirisen"

# 5.3 registrer deg selv som bruke
brukere["sirisoll"] = "Siri Sollerud"

# 5.4 skriv ut hanjo sitt navn
print(brukere["hanjo"])

# 5.5 fjern Hanne Johansen fra ordboken
brukere.pop("hanjo")
```

Oppgave 6: forbedre koden

```
def f1() :
     i = float(input("input"))
     print(i*i)



def f2() :
     i = input()
     print("velkommen til in1000 ", i)


def f3(  )     :
     print (    "Hei!"      )

f3()
f2()
f1()
```

- Generelt dårlig funksjonsnavn og variabelnavn, lite beskrivende
- f1 kan vi kalle “multipliser”
  - Gi bedre beskjed i terminalen til bruker om hva input skal være (ett tall)
- f2 kan vi kalle velkommen().
  - Endre variabelnavn i til navn
  - Gi bedre beskjed i terminalen til bruker om hva input skal være (et navn)
- f3 kan vi kalle hei():
  - Mange unødvendige mellomrom inni parantesen, disse syns ikke i utskriften uansett.

KONTAKT



...

sirisoll@uio.no
@sirisoll på Mattermost

UKENS MESTERVERK: “Cheese”

- ***Picasso pt. 137***