# GRUPPETIME UKE 4

**IN1000 GRUPPE 4 - 15.02.22**

# PLAN FOR GRUPPETIMEN

- Oblig 2 - hvordan gikk det?
- Jobbe sammen i Breakout Rooms
- Løkker
- Kombinere løkker og samlinger
- Parametre i prosedyrer
- Retur-verdier → funksjoner

# LÆRDOMMER FRA OBLIG 2

- Kodeflyt
  - Når programmet kjøres - hvilken rekkefølge utføres kodelinjene i gitt x antagelse(r)?
  - Se på Trix. 02.14 eller bruk pythontutor ETTER du har prøvd å resonnere deg frem selv først
- Kjører programmet?
  - Svaret kan være både ja og nei gitt forskjellige antagelser om hvordan programmet kjøres!
- Feilmeldinger
  - Se på understanding error messages
- Unngå laaange kommentarer
  - prøv å ikke overskrid hjelpestreken i Atom på høyre side
- Noen som prøvde seg på å forkorte temp til 2 desimaltall?

# 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*

# LÆRINGSMÅL UKE 4

- Kjenne til skrivemåte for while-løkker og for-løkker
- Kunne bruke løkker for å løse problemer
- Kunne bruke løkker sammen med samlinger
- Ha kjennskap til parametre i prosedyrer og parameteroverføring
- Kjenne til funksjoner og kunne bruke de for å få unngå redundans og få mer strukturerte programmer

# WHILE LØKKER

- Er det sant? → Ja, gå inn og kjør kodeblokk → Er det sant? → Nei, gå videre

```
x = 5     # kan også endre slik at x er input fra bruker
i = 0     # i er telleren vår, veldig vanlig å bruke
          # variabelnavnet i for teller i løkker.

while i < x:       # kan leses som "så lenge i er mindre enn 5"
    print(i)       # skriver ut telleren vår
    i = i + 1      # deretter øker vi variabelen i med 1
                   # kan alternativt skrives som i += 1
                   # så gjentas løkken
```

*→ Usikker hvordan koden kjøres? Prøv PythonTutor!*

# WHILE LØKKER

- I en while-løkke må det boolske uttrykket være modifiserbart
  - Dvs. at det bli evaluert til false på et eller annet tidspunkt, ellers så vil vi havne i en evig løkke
- Nyttig bruk for while-løkke er for eksempel en kommandoløkke, der man ikke avslutter før brukeren taster inn en spesifikk kommando.

```
i = 3
while i < 5:
    print(i)
```

*pass på å ikke lag evige løkker!*

# JOBB SAMMEN I BREAKOUT ROOMS!

→ uke 4 oppgaver

# WHILE-LOOP: gjenta til False

```
tekst = ["hadet", "på", "badet", "din", "gamle",
"sjokolade"]
indeks = 0

while indeks < len(tekst):
    print(tekst[indeks])
    indeks += 2
>> hadet
>> badet
>> gamle
```

*→ Altså, indekseringen øker med 2 for hver runde.*

while_loop.py

# WHILE-LOOP: gjenta til False

```
item_list = []
item = input("Add item or press 's' to stop: ")
while item != "s":
    item_list.append(item)
    item = input("Add item or press 's' to stop: ")
print("Your items are:", item_list)
```

→ *while loops er nyttige når vi ikke vet hvor mange ganger noe skal skje!*

# FOR LØKKER

- *for element in collection:*

  *do this codeblock*
- Kan bruke range() for å iterere x antall ganger
  - `for i in range(3)` `# Kjører 3 ganger → i er 0, 1, 2`
  - `for i in range(2, 5)` `# Kjører 3 ganger → i er 2, 3, 4`
  - `for i in range (2, 15, 3)` `# Kjører 5 ganger → i er 2, 5, 8, 11, 14`
- Brukes mest for å **iterere gjennom samlinger**

for_loop.py

# FOR-LOOP: iterere gjennom en samling

```
# Hva skrives ut?
div = ["kamera", "lommebok", "pass",
"mobillader"]
for ting in div:
     print("Pakket med:", ting)
>> Pakket med kamera
>> Pakket med lommebok
>> Pakket med pass
>> Pakket med mobillader
```

for_loop.py

# FOR-LOOP: iterere gjennom en samling

```
# Hva skrives ut?
div = ["kamera", "lommebok", "pass",
"mobillader"]
mat = ["pizza", "eple", "sjokolade", "sushi"]

for i in range(len(div)):
   div[i] = mat[i]
print(div)
```

```
>> ['pizza', 'eple', 'sjokolade', 'sushi']
```

# FOR VS. WHILE

For brukes gjerne når man vet hvor mange ganger noe skal skje, feks:

- range()
- Iterere gjennom en samling

While kan gjerne brukes når man ikke vet hvor mange ganger noe skal skje, men feks.

- Hvis man vil lagre et ukjent antall verdier en bruker taster helt til brukeren taster 0, da er det fordelaktig å bruke while.

| | |
|---|---|
| **Funksjon** | En funksjon som defineres med *def*, som ikke er del av en klasse (ordet *self* brukes ikke). **Funksjoner har alltid en returverdi.**<br>`def sum(a, b):`<br>`    c = a + b`<br>`    return c`<br>På engelsk kalles dette *function* |
| **Prosedyre** | Tilsvarende funksjon, men **uten returverdi**.<br>Bruker aldri ordet *self*, og er ikke del av en klasse.<br>`def skriv_ut(ord):`<br>`    print("ordet er", ord)`<br>På engelsk kalles dette *procedure* |
| **Metode** | **Tilsvarende funksjon, men som del av en klasse**. Har alltid *self* som første parameter. Kan, men må ikke, ha en returverdi.<br>`def areal(self):`<br>`    firkant_areal = self.lengde * self.bredde`<br>`    return firkant_areal`<br>På engelsk kalles dette *method* |

# MED PARAMETERE

- Til prosedyrer/funksjoner kan man også sende med parametre, som er en type variabler
  - Variablene man sender med “blir” de som er med som argument i prosedyren
- *def prosedyre_navn(parameter1, parameter2, …)*
  *gjør ting*
- ```
  def summer(a, b):
      print("Sum: ", a + b)
  x = 3
  y = 2
  summer(x, y)
  ```

*NB! Når prosedyren summer er ferdig så vil ikke a og b “beholde” verdiene sine. Fordel med prosedyrer -> kan gjenbruke kode! Kan kalle på en prosedyre flere ganger*
  ```

madlibs.py

# HVA SKRIVES UT?

```
def skriv_historie(forst, andre, tredje) :
    print(forst, "dro på ferie med ", tredje,
          "de ville dra uten", andre, "men",
          andre, "snek seg med i  bagasjerommet.." )


navn1 = "Silje"
navn2 = "Ole"
navn3 = "Jakob"
navne_liste = ["Emilie", "Håkon", "Yulai"]

skriv_historie("Kari", "Per", "Martin")
skriv_historie(navne_liste[0], navn3, navn1)
skriv_historie(navn2 + navn3, navne_liste[1],navne_liste[2])
```

```
>>Kari dro på ferie med Martin
de ville dra uten Per, men Per
snek seg med i bagasjerommet..

>>Emilie dro på ferie med Silje
de ville dra uten Jakob, men Jakob
snek seg med i bagasjerommet..

>>OleJakob dro på ferie med Yulai de
ville dra uten Håkon, men Håkon snek
seg med i bagasjerommet..
```

# FUNKSJONER

```
def summer(a, b):
    return a + b
    print("Dette skrives ikke ut!")

x = 3
y = 2
total = summer(x, y)
```

*Når vi skriver "return" så "avsluttes" funksjonen.*
*Videre linjer i funksjonen kjøres da ikke!*

*Til slutt vil total inneholde verdien 5,*
*fordi funksjonen summer(a, b) returnerer summen*
*av parameterne a + b*

*→ Kjekt å vite: skop*

# FUNKSJONER

```
# Hva printes hvis brukeren skriver inn 15?
def kontroll(alder):
    if alder >= 18:
        return "myndig"
    return "ikke myndig"

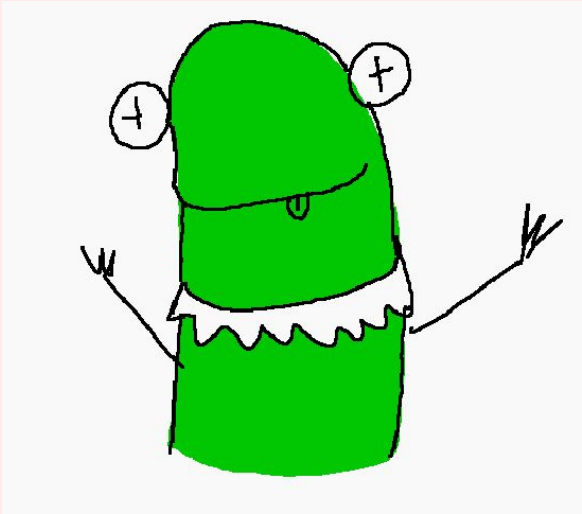
din_alder = int(input("Hvor gammel er
du?"))
svar = kontroll(din_alder)
print("Du er", svar)
```

# KONTAKT



...

sirisoll@uio.no
@sirisoll på Mattermost



# UKENS MESTERVERK: k3rmitz