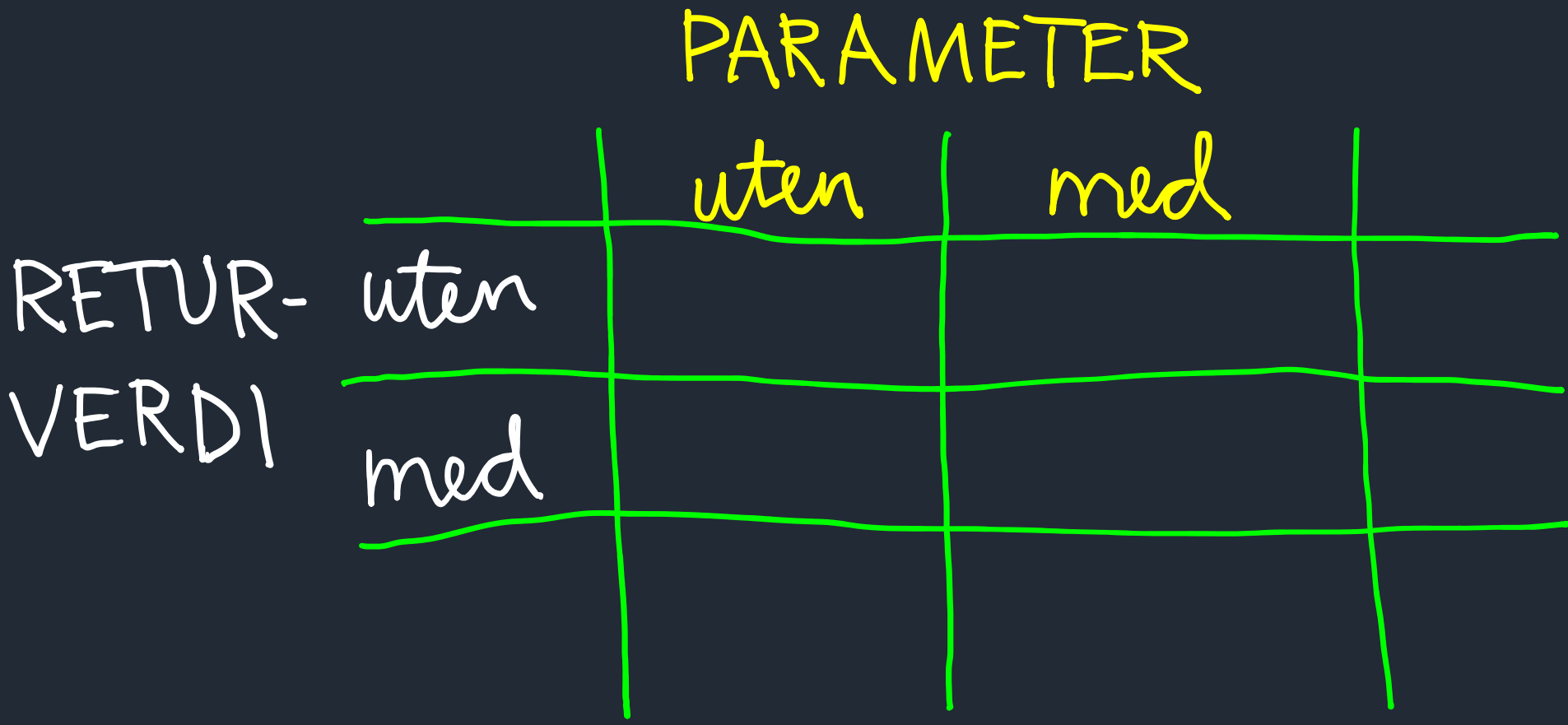
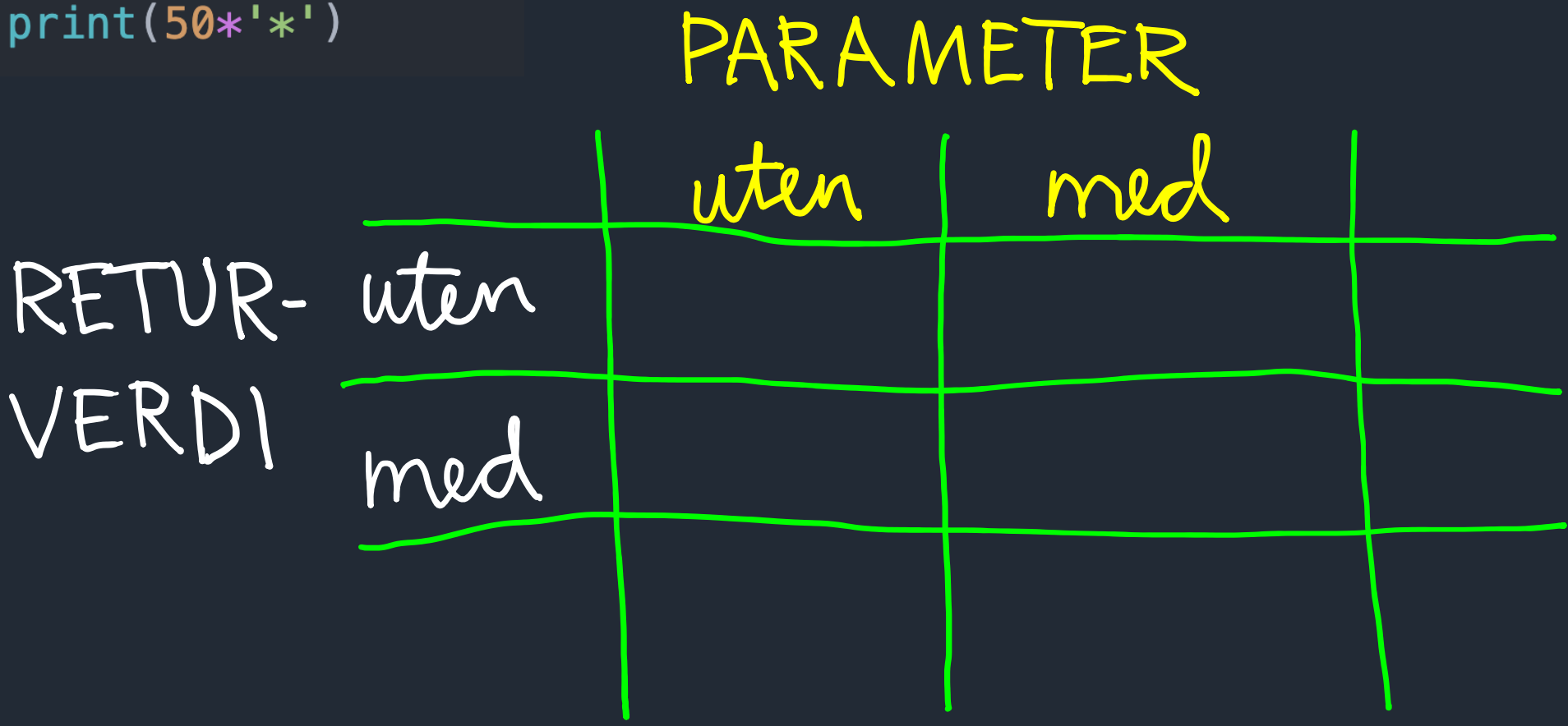
IN1000

Funksjoner og parametre

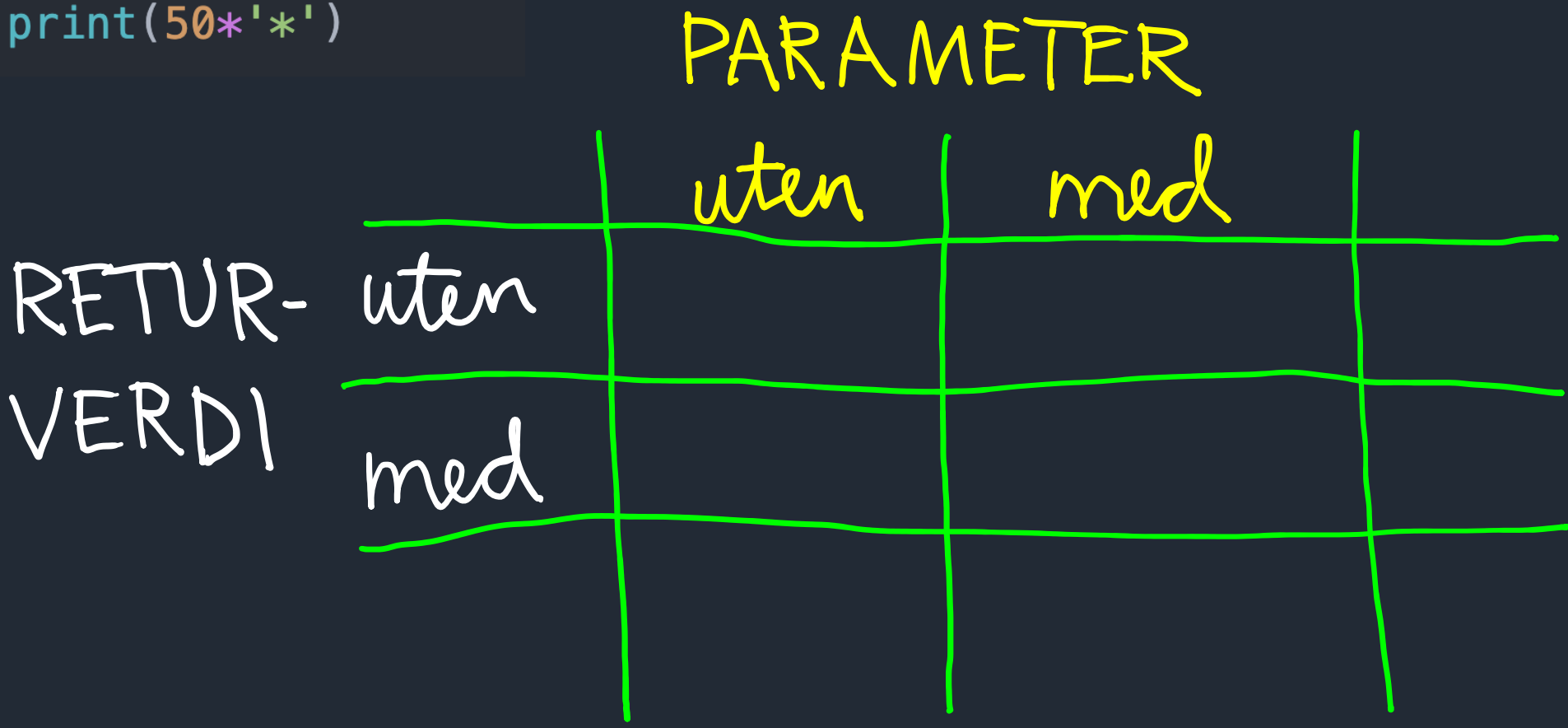

PARAMETER
uten
med
RETUR-
VERDI
uten
med

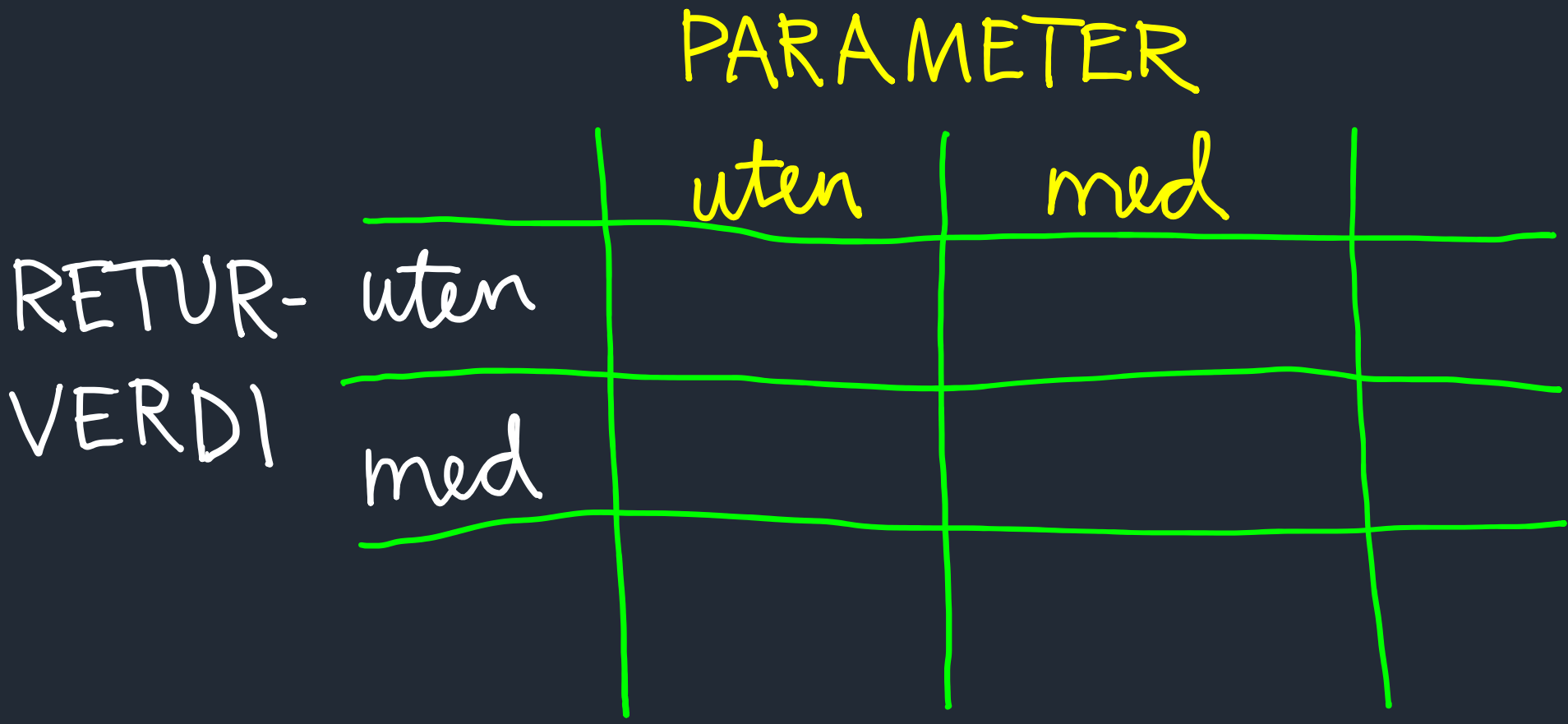
```python
def stjernelinje():
    print(50*'*')
```



```python
def stjernelinje():
    print(50*'*')
```

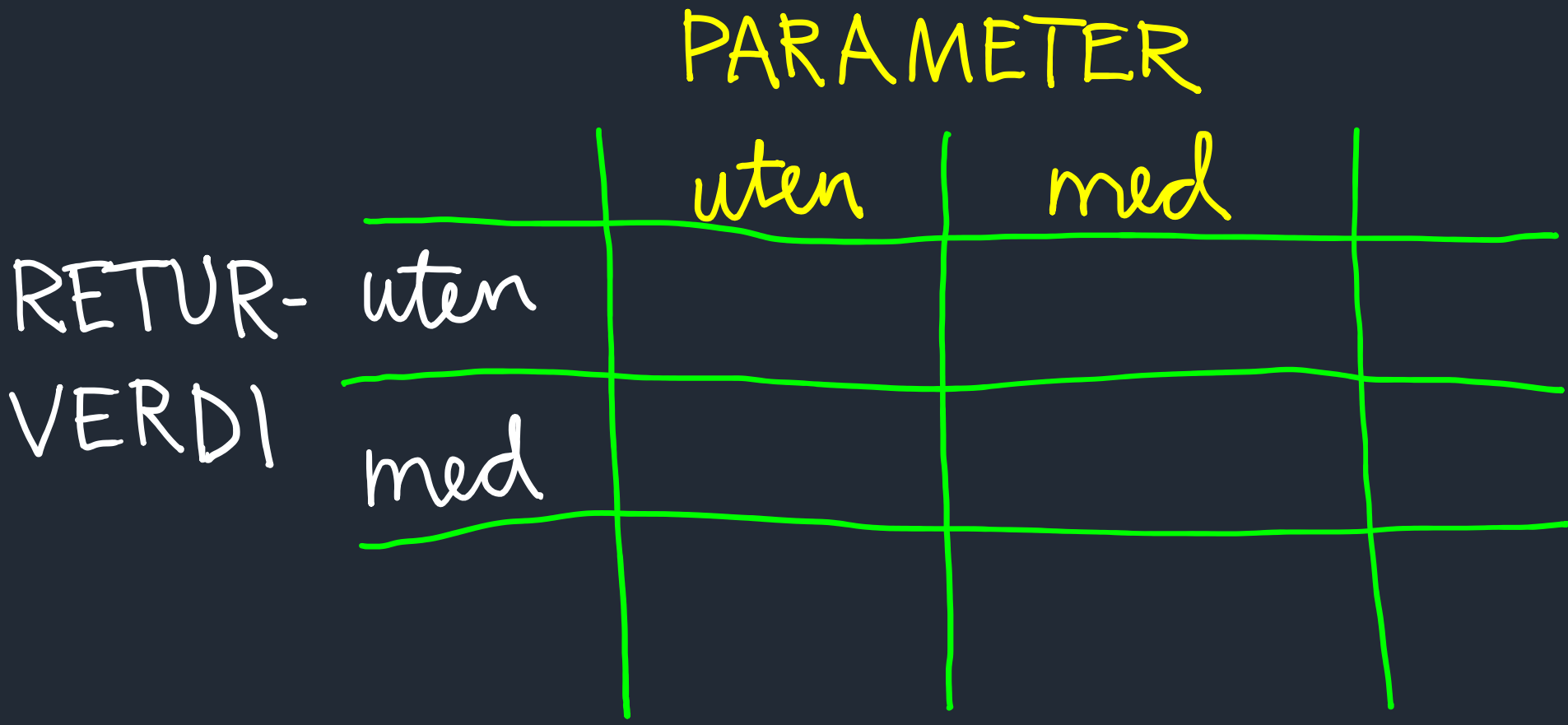
```python
print("Hei " + navn +"!")
```






```python
def alfabetetSomListe():
    abc = "ABCDEFGHIJKLMNOPQRSTUVWXYZÆØÅ"
    return list(abc)

```



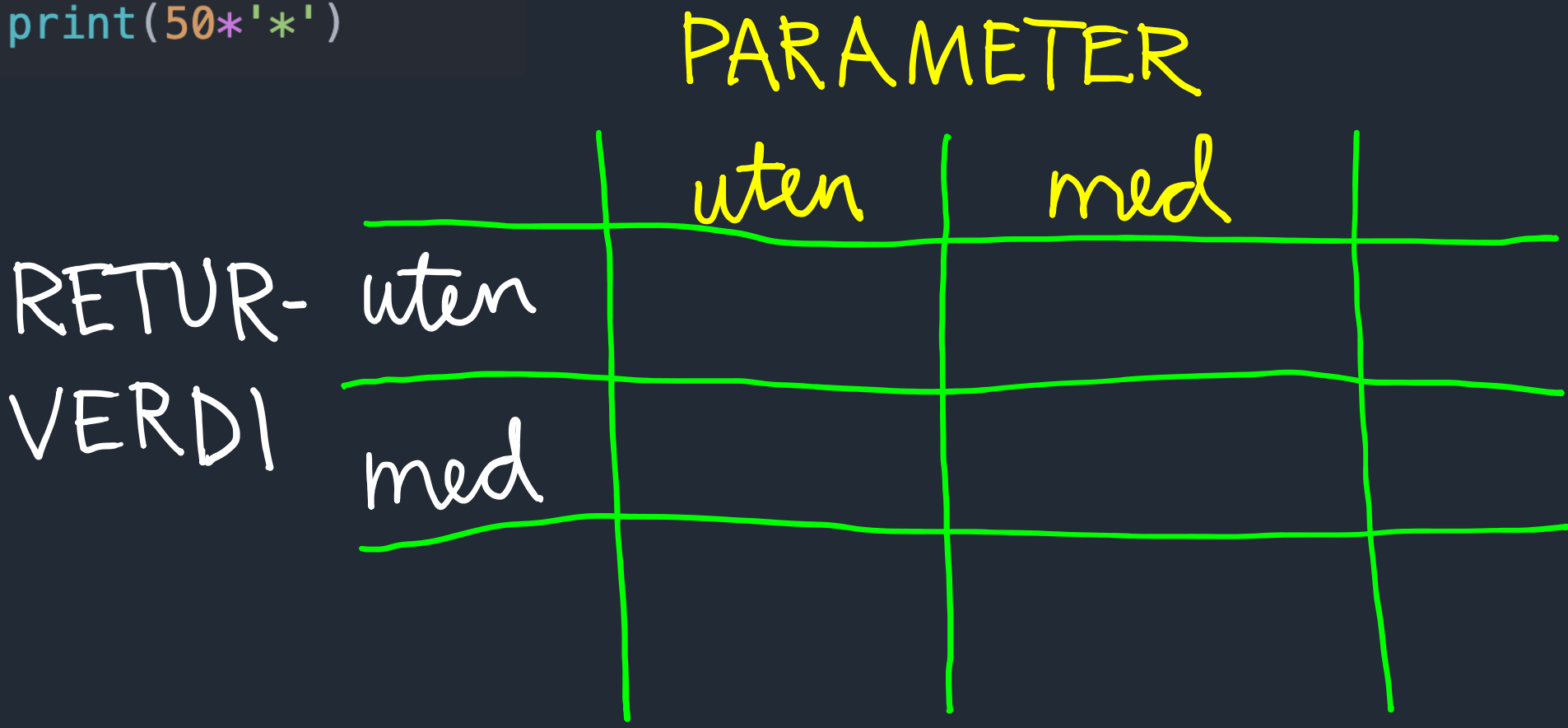

```python
def alfabetetSomListe():
    abc = "ABCDEFGHIJKLMNOPQRSTUVWXYZÆØÅ"
    return list(abc)

```

```python
lengde = len(fylkerAlfabetisk)
```

```python
def stjernelinje():
    print(50*'*')
```

```python
print("Hei " + navn +"!")
```



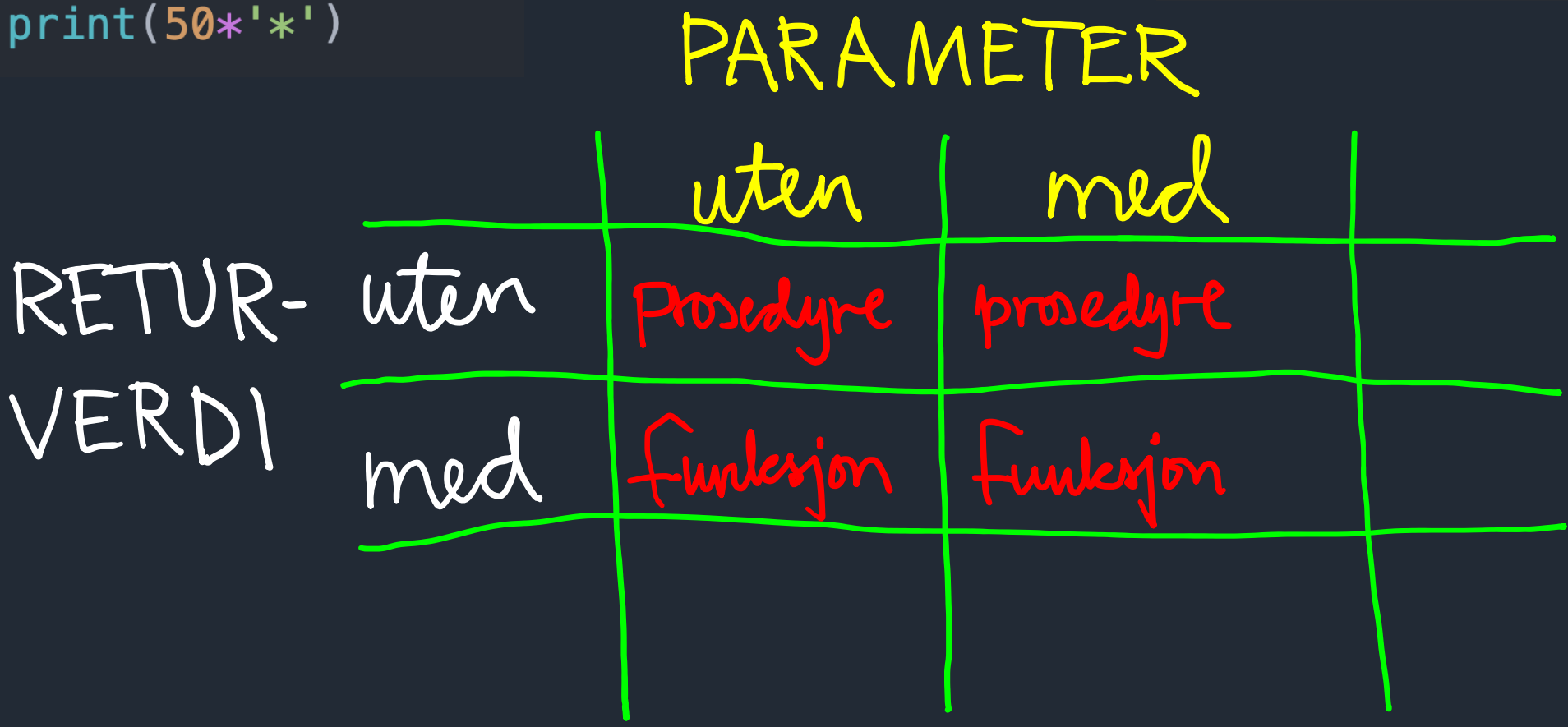

```python
def alfabetetSomListe():
    abc = "ABCDEFGHIJKLMNOPQRSTUVWXYZÆØÅ"
    return list(abc)

```

```python
lengde = len(fylkerAlfabetisk)
```

```
def stjernelinje():
    print(50*'*')
```

```
print("Hei " + navn +"!")
```




```
1  def alfabetetSomListe():
2      abc = "ABCDEFGHIJKLMNOPQRSTUVWXYZÆØÅ"
3      return list(abc)
4
```

```
lengde = len(fylkerAlfabetisk)
```

```
def minProsedyre1() :
        ...

def minProsedyre2(param1, param2, ....)
        ...

def minFunksjon1() :
        ...
        return verdi

def minFunksjon2(param1, param2, ....)
        ...
        return verdi
```

```
def skiltvakt() :
    svar = ""
    while svar != "ja" and svar != "JA" and svar != "Ja" :
        svar = input("Skriv 'ja' for å fortsette: ")

def jaEllerNei1() :
    svar = ""
    while svar != "ja" and svar != "nei" :
        svar = input("Skriv 'ja' eller 'nei': ")
    return svar

def jaEllerNei2(tekst) :
    svar = ""
    prompt = tekst + ": "
    while svar != "ja" and svar != "nei" :
        svar = input(prompt)
    return svar

print("1. Velkomen! Men før du kommer videre, må du skrive 'ja'!")
skiltvakt()
print("2. Takk for det. Da kan vi gå videre....")
svar = jaEllerNei1()
print("3. Du skrev", svar)
svar = jaEllerNei2("4. Du må svare enten 'ja' eller 'nei'")
print("5. Takk for nå")
```

```
print("'"+ord+"'", "oversatt til engelsk er",
      "'"+oversett(ord, franskeOrd, engelskeOrd)+"'")
```

Hvor mange argumenter har vi i dette kallet?

Hvor mange argumenter har vi i dette kallet?

Hvor mange argumenter har vi i dette kallet?

```
argument1 = ""+ord+""
argument2 = "oversatt til engelsk er"
argument3 = ""+oversett(ord, franskeOrd, engelskeOrd)+"")
```

```
print(""+ord+"", "oversatt til engelsk er",
      ""+oversett(ord, franskeOrd, engelskeOrd)+"")
```

Hvor mange argumenter har vi i dette kallet?

```
argument1 = ""+ord+""
argument2 = "oversatt til engelsk er"
argument3 = ""+oversett(ord, franskeOrd, engelskeOrd)+"")
```

- Parametre i prosedyrekallet kalles **argumenter**

Hvor mange argumenter har vi i dette kallet?

```
argument1 = ""+ord+""
argument2 = "oversatt til engelsk er"
argument3 = ""+oversett(ord, franskeOrd, engelskeOrd)+"")
```

- Parametre i prosedyrekallet kalles **argumenter**
- Argumenter er *uttrykk* med én verdi

Hvor mange argumenter har vi i dette kallet?

```
argument1 = ""+ord+""
argument2 = "oversatt til engelsk er"
argument3 = ""+oversett(ord, franskeOrd, engelskeOrd)+"")
```

- Parametre i prosedyrekallet kalles **argumenter**
- Argumenter er *uttrykk* med én verdi
- Det samme som det som kan stå til høyre for **=** i en tilordning

Vi har allerede brukt funksjoner med én aktuell parameter:

```
gruppeid = input("Oppgi en gruppe: ")

if gruppeid == "gruppe 1" :
```

input har ingen eller én parameter. Hvis vi har behov for å sette sammen en mer komplisert aktuell parameter, kan vi bruke '+' (tekstskjøtingsoperator), eller vi kan bygge opp en tekstvariabel som vi bruker som aktuell parameter:

```
input("Skriv inn " + str(antall) + " tegn adskilt med " + skilletegn + ": ")

prompt="Skriv inn " + str(antall) + " tegn adskilt med " + skilletegn + ": "
input(prompt)
```

```
def minFunksjon(parameter1,…):
    kodelinje1
    kodelinje2
    …
    return beregnet_verdi
```

```
def minFunksjon(parameter1,…):
    kodelinje1
    kodelinje2
    …
    return beregnet_verdi
```

For å kjøre alle kodelinjene i funksjonen ("kalle funksjonen"):

```
verdien_jeg_trenger = minFunksjons(argument1,…)
```

```
def minFunksjon(parameter1,…):
    kodelinje1
    kodelinje2
    …
    return beregnet_verdi
```

For å kjøre alle kodelinjene i funksjonen ("kalle funksjonen"):

argumenter

```
verdien_jeg_trenger = minFunksjons(argument1,…)
```

(Formelle) parametre

```
def minFunksjon(parameter1,…):
    kodelinje1
    kodelinje2
    …
    return beregnet_verdi
```

For å kjøre alle kodelinjene i funksjonen ("kalle funksjonen"):

argumenter

```
verdien_jeg_trenger = minFunksjons(argument1,…)
```

parametre

```
def minFunksjon(parameter1,…):
    kodelinje1
    kodelinje2
    …
    return beregnet_verdi
```

For å kjøre alle kodelinjene i funksjonen ("kalle funksjonen"):

argumenter

```
verdien_jeg_trenger = minFunksjons(argument1,…)
```

parametre

```
def minFunksjon(parameter1,…):
    kodelinje1
    kodelinje2
    …
    return beregnet_verdi
```

Parametrene er variabelnavn som er lokale variable i funksjonen. De får sin verdi fra verdiene til argumentene der funksjonen er kalt.

Argumentene er uttrykk, dvs. hvert argument evaluerer til én verdi, akkurat som i en variabeltilordning.

argumenter

```
verdien_jeg_trenger = minFunksjons(argument1,…)
```

```python
def gang_med_2(tall):
    return tall*2

def multipliser(a, b):
    return a*b

def potens(tall, eksponent):
    svar = 1
    for i in range(eksponent):
        svar = svar*tall
    return svar

def lag_velkomst(fag, person):
    return "Velkommen til " + fag + " kjære " + person

a = 3
print(gang_med_2(a))
print(gang_med_2(4+5*2))
print(potens(3,4))
print(lag_velkomst("IN1000", "Per Gynt"))
```

```python
def gang_med_2(tall):
    return tall*2

def multipliser(a, b):
    return a*b

def potens(tall, eksponent):
    svar = 1
    for i in range(eksponent):
        svar = svar*tall
    return svar

def lag_velkomst(fag, person):
    return "Velkommen til " + fag + " kjære " + person

a = 3
print(gang_med_2(a))
print(gang_med_2(4+5*2))
print(potens(3,4))
print(lag_velkomst("IN1000", "Per Gynt"))
```

```
2020_uke05 > python3 funksjon01.py
6
28
81
Velkommen til IN1000 kjære Per Gynt
2020_uke05 >
```

```
def beOmEnTekstverdi(oppfordring):
    tekst = input(oppfordring)
    return tekst
```

```
ord = beOmEnTekstverdi("Hvilket ord vil du oversette? ")

# Går i løkke så lenge ordet som skrives ikke er "STOPP!"
while ord != "STOPP!":
    if ord in franskeOrd:
        # Oversetter fra fransk til engelsk
        print("'"+ord+"'", "oversatt til engelsk er",
              "'"+oversett(ord, franskeOrd, engelskeOrd)+"'")
    elif ord in engelskeOrd:
        # Oversetter fra engelsk til fransk
        print("'"+ord+"'", "oversatt til fransk er",
              "'"+oversett(ord, engelskeOrd, franskeOrd)+"'")
    else:
        # Ordet fantes ike i noen ordliste
        print("Finner ikke", ord, "i ordlistene")

    ord = beOmEnTekstverdi("Hvilket ord vil du oversette? ")

# Her er ord == "STOPP!"
print("Takk for nå")
```

```
# Funksjon som oversetter ord ved å lete i fraOrdliste og hvis det finnes,
# returnerer ordet med samme indeks i tilOrdliste
def oversett(ord, fraOrdliste, tilOrdliste):
    # forutsetter:
    # ord er en tekstverdi
    # fraOrdliste og tilOrdliste er lister av tekster
    # de to ordlistene er «parallelle», dvs. at for en indeks k,
    # så er fraOrdliste[k] oversettelsen av tilOrdliste[k] og omvendt
    if ord in fraOrdliste:
        # Ord er i fraOrdliste
        for i in range(len(fraOrdliste)):
            if fraOrdliste[i] == ord:
                oversattOrd = tilOrdliste[i]
    else:
        # Ord er ikke i fraOrdliste
        oversattOrd = "FINNES IKKE!"
    return oversattOrd
```

```
def oversett1(ord, fraOrdliste, tilOrdliste):

    if ord in fraOrdliste:
        # Ord er i fraOrdliste
        for i in range(len(fraOrdliste)):
            if fraOrdliste[i] == ord:
                oversattOrd = tilOrdliste[i]
    else:
        # Ord er ikke i fraOrdliste
        oversattOrd = "FINNES IKKE!"

    return oversattOrd
```

```
def oversett2(ord, fraOrdliste, tilOrdliste):

    if ord in fraOrdliste:
        # Ord er i fraOrdliste
        for i in range(len(fraOrdliste)):
            if fraOrdliste[i] == ord:
                return tilOrdliste[i]
    else:
        # Ord er ikke i fraOrdliste
        return "FINNES IKKE!"
```

```
def oversett3(ord, fraOrdliste, tilOrdliste):

    if ord in fraOrdliste:
        # Ord er i fraOrdliste
        return tilOrdliste[fraOrdliste.index(ord)]
    else:
        # Ord er ikke i fraOrdliste
        return "FINNES IKKE!"
```

Men hvordan takler programmet vårt ord som finnes i begge språk?

```
ord = beOmEnTekstverdi("Hvilket ord vil du oversette? ")

# Går i løkke så lenge ordet som skrives ikke er "STOPP!"
while ord != "STOPP!":
    if ord in franskeOrd:
        # Oversetter fra fransk til engelsk
        print("'"+ord+"'", "oversatt til engelsk er",
              "'"+oversett(ord, franskeOrd, engelskeOrd)+"'")
    elif ord in engelskeOrd:
        # Oversetter fra engelsk til fransk
        print("'"+ord+"'", "oversatt til fransk er",
              "'"+oversett(ord, engelskeOrd, franskeOrd)+"'")
    else:
        # Ordet fantes ike i noen ordliste
        print("Finner ikke", ord, "i ordlistene")

    ord = beOmEnTekstverdi("Hvilket ord vil du oversette? ")

# Her er ord == "STOPP!"
print("Takk for nå")
```

Men hvordan takler programmet vårt ord som finnes i begge språk?

```
# Funksjon som finner ord som er med i begge ordlistene
# returnerer lista over duplikater
# parametre: 2 lister av ord
# returnerer liste med ord som finnes både i liste1 og liste2
def duplikatliste(liste1, liste2):
    duplikater = []
    for ord in liste1:
        if ord in liste2:
            duplikater.append(ord)
    return duplikater
```

Løkker gjør at kodelinjer kan kjøres flere ganger

- F.eks. gjøre lignende type utregning på ulike verdier

Løkker gjør at kodelinjer kan kjøres flere ganger

- F.eks. gjøre lignende type utregning på ulike verdier

Løkker og lister jobber godt sammen

- Kan generere og oppsummere store mengder verdier

Løkker gjør at kodelinjer kan kjøres flere ganger

- F.eks. gjøre lignende type utregning på ulike verdier

Løkker og lister jobber godt sammen

- Kan generere og oppsummere store mengder verdier

Prosedyrer og funksjoner gjør livet behagelig

- Tillater å tenke overordnet først, og deretter ordne detaljene

Løkker gjør at kodelinjer kan kjøres flere ganger

- F.eks. gjøre lignende type utregning på ulike verdier

Løkker og lister jobber godt sammen

- Kan generere og oppsummere store mengder verdier

Prosedyrer og funksjoner gjør livet behagelig

- Tillater å tenke overordnet først, og deretter ordne detaljene

Dere har nå verktøykassen for å løse skikkelige problemstillinger!