

IN1000 - uke 2

Presis forståelse av programmering

Forrige uke

- Programmering er problemløsning
- Bruke variabler for å ta vare på verdier
- Feilmeldinger
- Innlesing fra tastatur
- Beslutninger (if) - avgjøre hvilken kode som kjøres

Essensen i programmering

- Fra forrige uke:
 - "Programming is all about problem solving. It requires creativity, ingenuity, and invention"
 - Poenget er at det tekniske ikke er det sentrale
- **Men:** før man kan utfolde seg kreativt, må man ha presis kontroll over detaljene
 - Tour de France handler ikke om hvem som best balanserer på sykkelen
 - .. men det hjelper lite for femåringen som skal lære å sykle og går i grusen hver tredje meter

Programmering og presishet

- Man kan få til ganske mye basert på en omtrentlig forståelse av hvordan programkode kjøres
 - Men for å løse komplekse problemer og være trygg på at løsninger fungerer trenger man en presis forståelse
 - På et universitetsstudium er det ikke rom for magi - vi må kunne presist forutse og forklare oppførselen til programkode
 - Forståelsen blir ikke mer presis enn språket - for å snakke presist om kode lærer vi også terminologi

Målet for i dag

- Forstå nøyaktig hva som skjer i programmer av typen vi skrev forrige uke
 - Hvordan (i hvilken rekkefølge) operasjoner helt detaljert utføres på én enkelt linje
 - Hvordan et program flyter fra linje til linje
 - Hvilke feil som kan oppstå ved kjøring, og hva feilene betyr

Plan for dagen

- Hvordan én enkelt linje utføres:
 - Datatyper
 - Evaluering av uttrykk og funksjoner
- Hvordan et helt program utføres:
 - Kodeflyt fra linje til linje
 - Prosedyrer
- Sjekke antagelser og tolke feilmeldinger

Noen formelle begrep

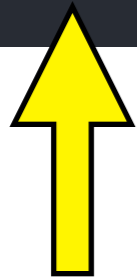
Programsetning (statement): *en linje i et Python-program*



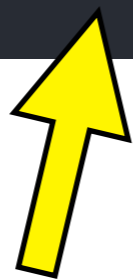
Uttrykk (expression): *noe som kan evalueres (beregnes) til en verdi*



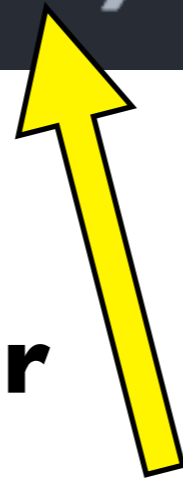
```
print(2+3)
```



**Biblioteks-
funksjon**



Operator



Literal

(av **datatypen** tall)

Plan for dagen

- Hvordan én enkelt linje utføres:
 - **Dat typer**
 - Evaluering av uttrykk og funksjoner
- Hvordan et helt program utføres:
 - Kodeflyt fra linje til linje
 - Prosedyrer
- Sjekke antagelser og tolke feilmeldinger

Datatyper

```
print("Hei")
```

Tekst (String)

```
print(5)
```

Heltall (Integer)

```
print(5.1)
```

Flyttall (Float)

```
print("5")
```

Tekst (pga hermetegn)

```
print('hei')
```

Enkle eller doble
hermetegn

Man kan sjekke datatype vha funksjonen *type*

```
type("Hei")  
type(5)  
type(5.1)  
type("5")  
type('hei')
```

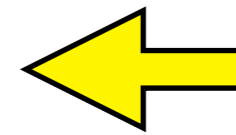
Operasjoner avhenger av datatype

```
print(2+3)
```



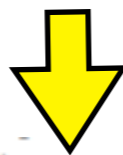
+ betyr addisjon

```
print("Hei" + "IN1000")
```



+ betyr
konkatenerer
tekst

```
print("Hei" + 1000)
```



```
guardian:kode_uke1 sandve$ python3 hei.py
```

```
Traceback (most recent call last):
```

```
File "hei.py", line 1, in <module>
```

```
print("Hei" + 1001)
```

```
TypeError: Can't convert 'int' object to str implicitly
```

Hva skrives ut her?

```
a = "5"  
b = "5"  
print(a + b)
```

Innhente tall fra brukeren

- Man ønsker ofte at bruker skal oppgi tall
 - **input** gir alltid en verdi av type tekst
- Biblioteksfunksjonen **int** konverterer en tekst til et tall (om mulig)
 - `tall = int("5")`
- Enkelt å teste ut:
 - `print("5"+"5")`
 - `print(int("5") + int("5"))`

Innhente tall fra brukeren

- Innhente tall:
 - Be brukeren skrive et tall
 - Bruke **input** til å hente det i form av tekst
 - Bruke **int** for å konvertere til heltall (integer)
 - Tilsvarende bruke **float** for å få flyttall (float)
 - (Man kan også konvertere tall til tekst med **str**)
- Om teksten ikke er tall får man feilmelding
 - Traceback (most recent call last):
File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: 'hei'
- [innlesing_tall.py]

Plan for dagen

- Hvordan én enkelt linje utføres:
 - Datatyper
 - **Evaluering av uttrykk og funksjoner**
- Hvordan et helt program utføres:
 - Kodeflyt fra linje til linje
 - Prosedyrer
- Sjekke antagelser og tolke feilmeldinger

Hva skjer innad på en linje?

- $\text{alder} = 6$
 - veldig rett frem..
- $\text{alder} = \text{alder} + 3$

Hva skjer innad på en linje?

- $\text{alder} = 6$
 - veldig rett frem..
- $\text{alder} = \text{alder} + 3$
 - Gjør ferdig høyresida for likhetstegnet først

Hva skjer innad på en linje?

- alder = 6
 - veldig rett frem..
- alder = $\overset{6}{\cancel{\text{alder}}} + 3$
 - Gjør ferdig høyresida for likhetstegnet først
 - Alle verdier er på høyresida slik de var før denne linja (alder er altså 6)

Hva skjer innad på en linje?

- alder = 6

- veldig rett frem..

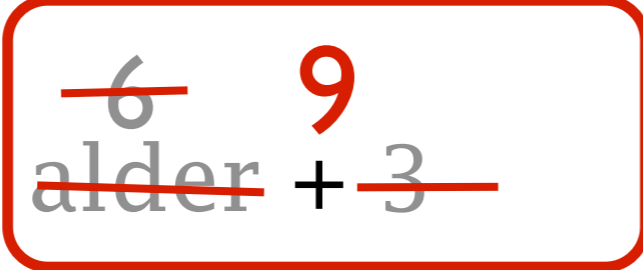
- alder = $\overline{\text{alder}} + \overline{3}$

- Gjør ferdig høyresida for likhetstegnet først
- Alle verdier er på høyresida slik de var før denne linja (alder er altså 6)
- Regner ut $6+3$ og får 9

Hva skjer innad på en linje?

- alder = 6

- veldig rett frem..

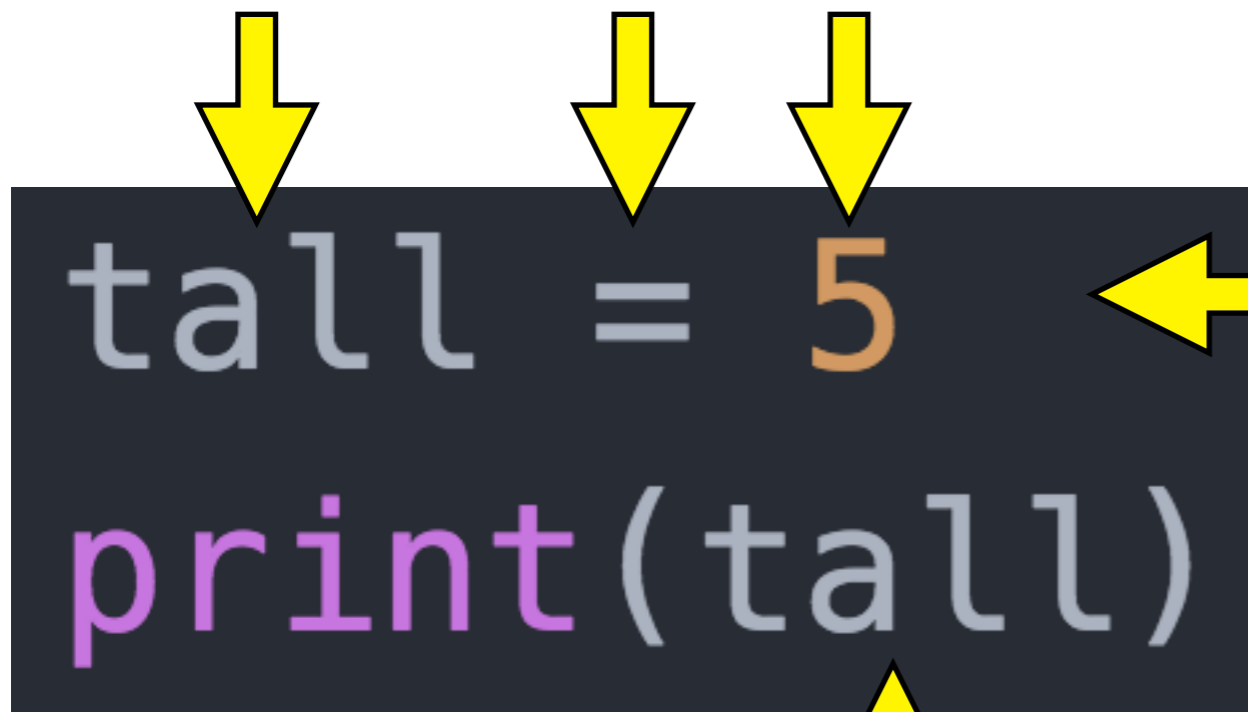
- alder =  $\overline{\text{alder} + 3}$

- Gjør ferdig høyresida for likhetstegnet først
- Alle verdier er på høyresida slik de var før denne linja (alder er altså 6)
- Regner ut $6+3$ og får 9
- Setter til slutt verdien 9 inn i alder

Formelt om tilordning av variabler

Tilordning:

variabelnavn = uttrykk



```
tall = 5  
print(tall)
```

Tilordning av variabel:
gir beskjed om at navnet tall
skal representere verdien 5

Bruk av variabel i et uttrykk:
henter frem verdien 5

Variabler:

settes fra og brukes i uttrykk

Tilordning:

variabelnavn = uttrykk

```
tall = 5+1  
print(tall+3)
```

Tilordning:
navnet tall gis
verdien av uttrykket (6)

Bruk av variabel i et uttrykk:

henter verdien 6, plusser med 3, og evaluerer dermed til 9

Variabler: verdien kan endres

```
tall = 5  
tall = 8  
print(tall)
```

variabelen tall gis verdien 5

verdien til tall vil fra nå av
(nedenfor) være 8

Bruk av variabel:
henter verdien 8

Variabler:

venstresiden versus høyresiden

```
tall = 5  
tall = tall + 2
```

variabelen tall
gis verdien 5

verdien til tall vil
fra nå av være 7

Venstreside
tilordner fremtidige
fra og med neste
tall ha verdien fra
høyresiden (**7**)

Ikke "**er lik**" som i
ligninger på ungdomsskolen,
men "**gjør lik**" som i
lignelser i bibelen
(vi sier "settes lik")

Høyreside
uttrykk:
"verdi -
hadde før
njen (**5**)

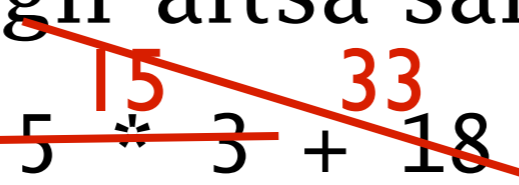
Evaluering av uttrykk

- Visse operasjoner gjøres alltid før andre
 - Python gjør f.eks.uansett ganging før plussing
- Følgende gir altså samme resultat 33:
 - ~~alder = 5 * 3 + 18~~
15 33

Evaluering av uttrykk

- Visse operasjoner gjøres alltid før andre
 - Python gjør f.eks.uansett ganging før plussing
- Følgende gir altså samme resultat 33:
 - ~~$alder = 5 * 3 + 18$~~
15 33
 - $alder = 18 + 5 * 3$

Evaluering av uttrykk

- Visse operasjoner gjøres alltid før andre
 - Python gjør f.eks.uansett ganging før plussing
- Følgende gir altså samme resultat 33:
 - ~~$alder = 5 * 3 + 18$~~

 - $alder = 18 + 5 * 3$
- Blir tydeligere med paranteser, bruk det!
 - $alder = (5 * 3) + 18$

Evaluering av uttrykk (forts.)

- For noen formål må man uansett ha paranteser

- $\text{alder} = \frac{5}{(3+2)} * 3 + 18$

Evaluering av uttrykk (forts.)

- For noen formål må man uansett ha paranteser

- $\text{alder} = \overbrace{(3+2)}^{-5} * \overbrace{3}^{15} + \overbrace{18}^{33}$

- Og for å gjøre det samme tydeligere - nøsting av paranteser er definitivt lovlig:

- $\text{alder} = ((3+2) * 3) + 18$

Evaluering av uttrykk (forts.)

- For noen formål må man uansett ha paranteser

- $\text{alder} = \overbrace{(3+2) * 3 + 18}^{\text{5} \quad \text{15} \quad \text{33}}$

- Og for å gjøre det samme tydeligere - nøsting av paranteser er definitivt lovlig:

- $\text{alder} = ((3+2) * 3) + 18$

- I praksis er det ikke tall man putter inn på slik måte, men variabler:

- $\text{alder} = ((\text{bachelor} + \text{master}) * \text{antallFagfelt}) + \text{barndom}$

En liten oppgave om variable og uttrykk

(fra eksamen 2016)

- **Oppgave 1a:**

Hva er verdien til **tall** etter at følgende kode er utført?

```
tall = 4 + (3 * 2)
```

```
tall = tall - 1
```

Hva er en condition?

if condition:
Statements

- **condition** er et boolsk uttrykk (*boolean expression*)
 - Noe som er sant eller ikke sant (**True** eller **False**)
 - Mer presist:
Et *uttrykk* som *evaluerer* til enten *verdien* **True** eller *verdien* **False**
 - Uttrykket kan være sammensatt

*"Just one condition you go to sleep right now:
That you don't touch my daughter
and in the morning, milk the cow"*
(Bob Dylan- Motorpsycho Nightmare)

Boolske uttrykk og verdier (Sannhetsverdier)

- Grunnleggende operasjoner: `<` `>` `==` `!=` `>=` `<=`
 - Kan brukes på tall (`5>3`) og på tekst ("`hei`" `==` "`hei`")
 - Slike uttrykk evaluerer til en verdi av typen **bool**
 - Typen bool har kun to mulige verdier: **True**, **False**
- Kan kombinere sannhetsverdier: `and`, `or`, `not`
 - `5>3 and 8<4`
 - `1>2 or 99>11`
 - `not 3>5`

Boolske uttrykk (forts)

- Utrykk kan naturligvis inneholde ulike variabler
 - `if alder>80 or dager_til_termin<10:`
`print("Ta mitt sete!")`
- Og så kan uttrykkene være nøstede
 - `if (dagerTilTermin>180 and tidspunkt=="morgen")`
`or (alder<6 and antallKilometerBiltur>30):`
`print("du er sikkert kvalm!")`

Boolske variable

- Boolske verdier kan holdes på av variabler (på samme måte som for tall og tekst)
- En boolsk variabel kan brukes alle steder hvor man kan bruke en boolsk verdi

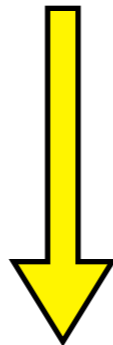
```
betalerHalvPris = alder<18 or alder>66  
if betalerHalvPris:  
    ...
```

Evaluering av funksjoner

- En viktig del av programmering (*som vi kommer mer tilbake til senere*) er ***funksjoner***
 - Kan være biblioteksfunksjoner eller egne
 - Vi har f.eks. brukt *input* som henter en verdi fra brukeren (tastaturet)

Hvordan **input** egentlig virker

- 1: **input** fryser programmet inntil brukeren skriver et svar
- 2: deretter *evaluerer* **input** til en *verdi* av typen tekst (str)
(det brukeren skrev inn)



```
navn = input()
```



Verdien (som brukeren skrev inn)
tilordnes til variabelen navn

Bruk av en funksjon evaluerer til en verdi

```
tall = int("1" + "3")
```



evaluerer til "13"
(og sendes til int)



evaluerer til 13
(av type tall)



verdien 13 tilordnes variabelen tall

Bruk av funksjoner kan også nøstes

```
tall = int( input() )
```



input evaluerer til f.eks. "13"
(som sendes videre til int)



int evaluerer til 13



verdien 13 tilordnes variabelen tall

Hva er galt her?

Oppgave:

Les inn en fart. Dersom fart er 60 eller mindre, skal man lage en streng som består av "fart:" og selve farten (f.eks. "fart:56"). Dersom fart er høyere enn 60, skal man lage strengen "fart:over 60".

```
fart = int(input())
if fart <= 60:
    svar1 = print("Fart: " + fart)
else:
    svar2 = print("Fart: over 60.")
```


Hva er galt her?

Oppgave:

Les inn en fart. Dersom fart er 60 eller mindre, skal man lage en streng som består av "fart:" og selve farten (f.eks. "fart:56"). Dersom fart er høyere enn 60, skal man lage strengen "fart:over 60".

```
fart = int(input(""))  
if fart <= 60:  
    svar1 = print("Fart: " + fart)  
else:  
    svar2 = print("Fart: over 60.")
```

svar1 blir tilordnet
det som print
evaluerer til
- ikke noe (None)

Hva er galt her?

Oppgave:

Les inn en fart. Dersom fart er 60 eller mindre, skal man lage en streng som består av "fart:" og selve farten (f.eks. "fart:56"). Dersom fart er høyere enn 60, skal man lage strengen "fart:over 60".

```
fart = int(input(""))
if fart <= 60:
    svar1 = print("Fart: " + fart)
else:
    svar2 = print("Fart: over 60.")
```

Nå har man enten en variabel svar1 eller svar2 - men har ikke kontroll på hvilken..

Endre det til å bli korrekt

Oppgave:

Les inn en fart. Dersom fart er 60 eller mindre, skal man lage en streng som består av "fart:" og selve farten (f.eks. "fart:56"). Dersom fart er høyere enn 60, skal skal man lage strengen "fart:over 60".

```
fart = int(input(""))
if fart <= 60:
    svar = "Fart: " + fart
else:
    svar = "Fart: over 60."
```

Endre det til å bli korrekt

Oppgave:

Les inn en fart. Dersom fart er 60 eller mindre, skal man lage en streng som består av "fart:" og selve farten (f.eks. "fart:56"). Dersom fart er høyere enn 60, skal man lage strengen "fart:over 60".

```
fart = int(input(""))
if fart <= 60:
    svar = "Fart: " + str(fart)
else:
    svar = "Fart: over 60."
```

Nøtt: hva skjer her?

Kode:

```
svar = input( input("Sporsmaal: ") )  
print("Du svarte: " + svar)
```

Kjøring:

```
Sporsmaal: hvilket studium?  
hvilket studium?informatikk  
Du svarte: informatikk
```

Nøtt: hva skjer her?

```
Sporsmaal: hvilket studium?  
hvilket studium?informatikk  
Du svarte: informatikk
```

```
svar = input( input("Sporsmaal: ") )
```

←→ skriver ut "Sporsmaal: "
evaluerer til "hvilket studium?"
som sendes til den ytre *input*

←→ skriver ut "hvilket studium?"
evaluerer til "informatikk"
som tilordnes variabelen *svar*

```
print("Du svarte: " + svar)
```

skriver ut "Du svarte: informatikk"

Nøtt: hva skjer her?

```
Sporsmaal: hvilket studium?  
hvilket studium?informatikk  
Du svarte: informatikk
```

```
svar = input( input("Sporsmaal: ") )
```



skriver ut "Sporsmaal: "
evaluerer til "hvilket studium?"
som sendes til den ytre *input*



skriver ut "hvilket studium?"
evaluerer til "informatikk"
som tilordnes variabelen *svar*

```
print("Du svarte: " + svar)
```

skriver ut "Du svarte: informatikk"

Prinsippene for evaluering av en programsetning

- Man begynner alltid innerst og beveger seg utover
 - $18 + ((3+2) * 3)$
 - `int(input())`
- Visse operasjoner utføres før andre
 - $3 + 2 * 3$
- I en tilordning bestemmer man høyresiden først
 - `alder = alder + 3`

Plan for dagen

- Hvordan én enkelt linje utføres:
 - Datatyper
 - Evaluering av uttrykk og funksjoner
- Hvordan et helt program utføres:
 - **Kodeflyt fra linje til linje**
 - Prosedyrer
- Sjekke antagelser og tolke feilmeldinger

Hvordan et program flytter fra linje til linje

- Dette er temmelig enkelt for det vi har lært frem til nå (endrer seg om et kvarter..)
- Hovedregel:
 - Gjør ferdig en linje, deretter gå til linjen nedenfor

Kodeflyt ved beslutninger (**if**)

- *if* **boolsk uttrykk**:
kodeblokk1
etterfølgende
- Siden et boolsk uttrykk kun kan evaluere til to mulige verdier:
 - Om uttrykket har verdien **True** gå til *kodeblokk1*
 - Om uttrykket har verdien **False** gå til *etterfølgende*

Kodeflyt ved beslutninger (**if-else**)

- *if boolsk uttrykk:*
kodeblokk1
else:
kodeblokk2
etterfølgende
- Siden et boolsk uttrykk kun kan evaluere til to mulige verdier:
 - Om uttrykket har verdien **True** gå til kodeblokk1
 - Om uttrykket har verdien **False** gå til kodeblokk2

Kodeflyt ved beslutninger (**elif**)

- *if* *boolsk uttrykk1*:
 kodeblokk1
elif *boolsk uttrykk2*:
 kodeblokk2
etterfølgende
- Om boolsk uttrykk1 har verdien **True** gå til kodeblokk1
- Om boolsk uttrykk1 har verdien **False** gå til elif
 - Om boolsk uttrykk2 har verdien **True** gå til kodeblokk2
 - Om boolsk uttrykk2 har verdien **False** gå til etterfølgende

En liten test på problemløsning

Skriv (med blyant og papir) en kode som finner
den minste av to verdier:


```
tall1 = int(input("Skriv tall 1: "))  
tall2 = int(input("Skriv tall 2: "))  
  
#skriv kode her som tilordner den minste  
#av verdiene tall1 og tall2 til variabelen minst  
  
print(minst)
```

{Mulig løsning: minst.py}

Etterlign kjøring, med blyant og papir

- Gjør manuelt det samme som datamaskinen ville gjort, linje for linje
 - Kan gjøres i hodet, men enklere på papir (print ut koden og bruk blyant)
 - Vær presis - her er hver detalj viktig

Følge et program, linje for linje

 lengde=7
bredde=4

```
if lengde==bredde:  
    omkrets = 4*lengde  
else:  
    omkrets = (2*lengde) + (2*bredde)  
  
print("Omkrets: " + str(omkrets))
```


Følge et program, linje for linje

```
lengde=7  
bredde=4
```

```
→ if 7 == bredde:  
    omkrets = 4*lengde  
else:  
    omkrets = (2*lengde) + (2*bredde)  
  
print("Omkrets: " + str(omkrets))
```

Følge et program, linje for linje

lengde=7
bredde=4

```
→ if 7 == 4 :  
    omkrets = 4*lengde  
    else:  
        omkrets = (2*lengde) + (2*bredde)  
  
print("Omkrets: " + str(omkrets))
```

Følge et program, linje for linje

```
lengde=7  
bredde=4
```

```
if 7 == 4 :  
omkrets = 4*lengde  
else:
```

```
→ omkrets = (2* 7 ) + (2*bredde)
```

```
print("Omkrets: " + str(omkrets))
```

Følge et program, linje for linje

```
lengde=7  
bredde=4
```

```
if 7 == 4 :  
omkrets = 4*lengde  
else:
```

```
→ omkrets = ( 14 ) + (2*bredde)
```

```
print("Omkrets: " + str(omkrets))
```

Følge et program, linje for linje

```
lengde=7  
bredde=4
```

```
if 7 == 4 :  
omkrets = 4*lengde  
else:
```

```
→ omkrets = ( 14 ) + (2* 4 )
```

```
print("Omkrets: " + str(omkrets))
```

Følge et program, linje for linje

```
lengde=7  
bredde=4
```

```
if 7 == 4 :  
    omkrets = 4*lengde  
else:
```

```
→ omkrets = ( 14 ) + ( 8 )
```

```
print("Omkrets: " + str(omkrets))
```

Følge et program, linje for linje

```
lengde=7  
bredde=4
```

```
if 7 == 4 :  
omkrets = 4*lengde  
else:
```

```
→ omkrets = ( 22 )
```

```
print("Omkrets: " + str(omkrets))
```

Følge et program, linje for linje

```
lengde=7  
bredde=4
```

```
if 7 == 4 :  
omkrets = 4*lengde  
else:  
    omkrets = ( 22 )
```

```
→ print("Omkrets: " + str( 22 ))
```


Følge et program, linje for linje

```
lengde=7  
bredde=4
```

```
if 7 == 4 :  
omkrets = 4*lengde  
else:  
    omkrets = ( 22 )
```

```
→ print("Omkrets: " + "22" )
```

Et litt mer vrient problem

- Spør brukeren om alder (*bruk input*):
 - Dersom mindre enn 6: skriv ut "Lek i skogen"
 - Dersom mindre enn 3: skriv ut "Lek i lekegrinda"
 - (ikke skriv ut noe ellers)
 - Skal uansett skrive ut maksimalt én setning
- Hvordan vil du nå skrive koden?
 - Prøv selv med blyant og papir! (3 minutt)
 - Etterpå diskuter med nabo (3 minutt)

Hvorfor blir følgende løsning feil?

```
if alder<3:  
    print("Lek i lekegrinda")
```

```
if alder<6:  
    print("Lek i skogen")
```

Løsning med kombinert uttrykk

```
if alder<3:  
    print("Lek i lekegrinda")
```

```
if alder<6 and alder>3:  
    print("Lek i skogen")
```

Løsning med else-if

```
if alder<3:  
    print("Lek i lekegrinda")  
else:  
    if alder<6:  
        print("Lek i skogen")
```

Løsning med elif

```
if alder<3:  
    print("Lek i lekegrinda")  
elif alder<6:  
    print("Lek i skogen")
```

Og hvorfor går ikke den motsatte elif?

```
if alder<6:  
    print("Lek i skogen")  
elif alder<3:  
    print("Lek i lekegrinda")
```

Løsning med nøsting

```
if alder<6:  
    if alder<3:  
        print("Lek i lekegrinda")  
    else:  
        print("Lek i skogen")
```


Plan for dagen

- Hvordan én enkelt linje utføres:
 - Datatyper
 - Evaluering av uttrykk og funksjoner
- Hvordan et helt program utføres:
 - Kodeflyt fra linje til linje
 - **Prosedyrer**
- Sjekke antagelser og tolke feilmeldinger

Vi trenger mer struktur!

- Vi har frem til nå skrevet programmer linje for linje nedover i en fil
- Realistiske program er imidlertid ofte tusener eller millioner av linjer!
 - Ingen kan ha oversikt over en flat liste på mange tusen (eller millioner) av linjer
- Et første nivå av strukturering er **subrutine**: en **navngitt blokk** med kodelinjer, som kan **kalles** og **tilpasses**

Ulike versjoner av subrutiner

- Subrutiner kommer i ulike versjoner, av gradvis økende kompleksitet
- Vi vil introdusere de involverte aspektene stegvis
 - I dag: Prosedyre - **uten** parametre og returverdi
 - Om to uker: Prosedyre - med **parametre**
 - Om to uker: Funksjon - med **returverdi**
 - Litt senere i høst: **Instans**-metode (OO)

Hvordan kan en prosedyre se ut

```
def mittProsedyreNavn():  
    kodelinje1  
    kodelinje2  
    ...
```

For å kjøre alle kodelinjene i prosedyren ("*kalle*
prosedyren"):

```
mittProsedyreNavn()
```

Eksempel på prosedyre: kodeblokk

```
print("Jeg sier dette bare en gang!")  
print("Da var jeg ferdig!")
```

Eksempel på prosedyre: navn

```
def giBeskjed():  
    print("Jeg sier dette bare en gang!")  
    print("Da var jeg ferdig!")
```

Eksempel på prosedyre: kall

```
def giBeskjed():  
    print("Jeg sier dette bare en gang!")  
    print("Da var jeg ferdig!")
```

giBeskjed()



Kontrollflyt og metoder

```
def giBeskjed()  
    print("Jeg sier dette bare en gang!")
```

```
def kallVidere()  
    print("Her kommer det")  
    giBeskjed()  
    print("Og en gang til")  
    giBeskjed()  
    print("Det var ikke mye")
```

```
print("Hei, jeg vil si deg noe")  
kallVidere()  
print("Fikk du det likevel med deg?")
```


Kontrollflyt og metoder

```
def giBeskjed()  
    print("Jeg sier dette bare en gang!")
```

```
def kallVidere()  
    print("Her kommer det")  
    ➔ giBeskjed()  
    print("Og en gang til")  
    ➔ giBeskjed()  
    print("Det var ikke mye")
```

```
➔ print("Hei, jeg vil si deg noe")  
➔ kallVidere()  
    print("Fikk du det likevel med deg?")
```

Kontrollflyt og metoder

```
def giBeskjed()  
    print("Jeg sier dette bare en gang!")
```

```
def kallVidere()  
    print("Her kommer det")  
    giBeskjed()  
    print("Og en gang til")  
    giBeskjed()  
    print("Det var ikke mye")
```

```
print("Hei, jeg vil si deg noe")  
kallVidere()  
print("Fikk du det likevel med deg?")
```

Hva skrives ut her?

```
def p1():  
    print("B")  
    print("C")
```

```
def p2():  
    p1()  
    print("A")
```

```
p1()  
p2()
```

Konklusjon

- *Programsetninger* består av *uttrykk* som evaluerer til *verdier* ifølge helt presise regler
- En verdi er alltid av en bestemt *datatype*, f.eks. heltall, flyttall, tekst eller boolsk verdi
- *Kodeflyt* er presist definert etter stort sett enkle regler (for påfølgende linjer, beslutninger, *prosedyrer* osv)