INF1001 - uke 2

Inputt, beslutninger,
kontrollflyt og prosedyrer

Outline:

- Repetisjon fra forrige uke
- Lese inn fra tastaturet
- Beslutninger
- Kontrollflyt
- Prosedyrer

Outline:

- Repetisjon fra forrige uke
- Lese inn fra tastaturet
- Beslutninger
- Kontrollflyt
- Prosedyrer

Repetisjon fra forrige uke

- Et program består av flere programsetninger (linjer) som kjøres etter hverandre:

```
print("Hei")
print("INF1001")
```

- Innad på en linje har man ofte et uttrykk - noe som kan evaluere til en verdi:
  - `2+3`
- En verdi er alltid av en bestemt type:
  - `int, float, str ...`

Repetisjon fra forrige uke

- En variabel kan holde på en verdi for senere bruk
  - Blir tilordnet en verdi: `tall = 5`
  - Verdien kan endres: `tall = 7`
  - Verdien kan senere brukes: `print(tall)`
  - Kan være på begge sider av likhetstegnet: `tall=tall+2`
- Man kan dobbeltsjekke antagelser med assert:

```
tall=(4+1)*2
assert tall=10
```

Repetisjon fra forrige uke

- Vanligvis skriver man et program i en editor, og kjører etterpå hele programmet
- I Python-tolken kan man kjøre enkeltlinjer, nyttig for å få presis forståelse

Outline:

- Repetisjon fra forrige uke
- Lese inn fra tastaturet
- Beslutninger
- Kontrollflyt
- Prosedyrer

Slipp brukeren til!

- De fleste program tar en eller annen jevnlig inputt fra brukeren
- Dette gir også mye mer dynamikk i programmene enn det vi har sett på til nå
- Kommunisere med brukeren av et program:
  - Biblioteksfunksjonen **`print`** gir output til bruker
  - Biblioteksfunksjonen **`input`** henter inputt fra bruker
- [innlesing.py]

Gi brukeren beskjed om å skrive inn et svar

- Brukeren må vite at han/hun skal skrive noe:

```
print("Hva heter du? ")
navn = input()
```

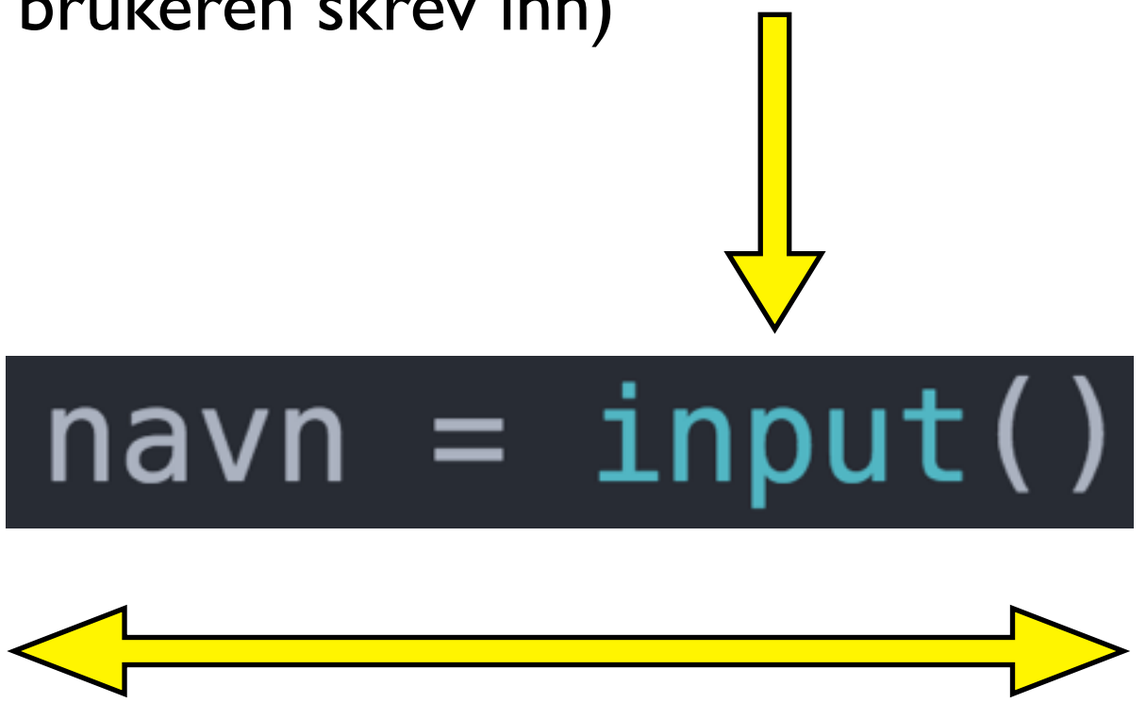
- **input** kan skrive beskjed før den venter på svar:

```
navn = input("Hva heter du? ")
```

Hvordan **input** egentlig virker

1: **input** fryser programmet inntil brukeren skriver et svar

2: deretter *evaluerer* **input** til en *verdi* av typen tekst
(det brukeren skrev inn)



Verdien (som brukeren skrev inn)
tilordnes til variabelen navn

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")
  - Test det gjerne ut:
    - assert "5"+"5" == "55"
    - assert int("5") + int("5") == 10

Innhente tall fra brukeren

- 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'
    ```
- 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)**
- [innlesing_tall.py]
    ```

Outline:

- Repetisjon fra forrige uke
- Lese inn fra tastaturet
- Beslutninger
- Kontrollflyt
- Prosedyrer

Ikke bare følge strømmen

- For det vi har sett på til nå,
  har programmet alltid fulgt én bestemt sekvens av instruksjoner
  - Vi kan lese ulike startverdier fra tastatur og dermed få ulike resultat, men alltid basert på de samme operasjonene
- Vi trenger mer variasjon!
  - Det blir mye artigere å programmere dersom hvilke operasjoner som utføres også kan avhenge av verdier man regner ut

Et konkret problem som krever beslutninger

- Problemstilling:
  - Vi ønsker å lage et program som spør brukeren om alder. Dersom brukeren er over 6 år, skal den skrive "Velkommen til mitt program". Dersom brukeren er 6 år eller mindre skal den skrive "Gå heller ut og lek i skogen".
- Vi trenger altså to ulike `print`
- Vi må imidlertid sørge for at kun én `print` blir kjørt, hvor hvilken som blir kjørt avhenger av innlest verdi for `alder`

En uferdig løsning (mangler beslutning)

```
innlest = input("Hvor gammel er du? ")
alder = int(innlest)

print("Velkommen til mitt program")
print("Gå heller ut og lek i skogen")
```

Beslutninger i et program: `if`

- Syntaks (skrivemåte):

```
if condition:
  Statement1
  Statement2
    ..
```

- NB! Korrekt innrykk er helt essensielt!
- Eksempel:

```
if alder>6:
  print("velkommen")
```

- {beslutning.py}

En praktisk kortform: `if-else`

- Syntaks:
  - if condition:
      Statement1
      Statement2
    else:
      Statement1
      Statement2

- {beslutning2.py}

Prøv en variant selv!

- Modifisert problemstilling:
  - Mindre enn 6: skriv "Lek i skogen"
  - Mindre enn 3: skriv "Lek i lekegrinda"
  - 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")
```

Hva er en condition?

"Just one condition you go to sleep right now:
That you don't touch my daughter
and in the morning, milk the cow"

```
if condition:
    Statements
```

- **condition** er et boolsk uttrykk *(boolean expression)*
  - Noe som er sant eller ikke sant (**True** eller **False**)
  - Vi så det allerede i første uke:
    `print(5>4)`
    `print(4>5)`
  - Vi så nettopp et mer komplekst eksempel:
    `if alder<6 and alder>3:`

Boolske uttrykk (Sannhentsverdier)

- 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 utrykkene være nøstede
  - if (dagerTilTermin>180 **and** tidspunkt=="morgen") **or (**alder<6 **and** antallKilometerBiltur>30**):**
    print("du er sikkert kvalm!")

Formell definisjon av **if**

- if boolean expression:
    Statements1
  else:
    Statements2

- Siden et boolsk uttrykk (boolean expression) kun kan ha to mulige verdier:
  - Om uttrykket har verdien **True** kjøres Statements1
  - Om uttrykket har verdien **False** kjøres Statements2

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:
   ...
```

En liten test på problemløsning

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

```
innlest1 = input("Skriv tall 1: ")
tall1 = int(innlest1)
innlest2 = input("Skriv tall 1: ")
tall2 = int(innlest2)

#skriv kode her som tilordner den minste
#av verdiene tall1 og tall2 til variabelen minst

assert minst <= tall1
assert minst <= tall2
print(minst)
```

{Mulig løsning: minst.py}

Outline:

- Repetisjon fra forrige uke
- Lese inn fra tastaturet
- Beslutninger
- Kontrollflyt
- Prosedyrer

Hvordan et program presist utføres

- Hvordan (i hvilken rekkefølge) utføres operasjoner helt detaljert på én enkelt linje
- Hvordan flyter (helt presist) et program fra linje til linje

Hva skjer innad på en linje?

- alder = 6
  - veldig rett frem..
- alder = alder + 3

Hva skjer innad på en linje?

- alder = 6
  - veldig rett frem..
- alder = alder + 3
  - Gjør ferdig høyresida for likhetstegnet først

Hva skjer innad på en linje?

- alder = 6
  - veldig rett frem..
- alder = 6 ~~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 = ~~6~~ 9 ~~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

Hva skjer innad på en linje?

- alder = 6
  - veldig rett frem..
- alder = ~~alder~~ + ~~3~~ (~~6~~ 9)
  - 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

Videre om 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

Videre om 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`

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

Videre om evaluering av uttrykk (forts.)

- For noen formål må man uansett ha paranteser
  - `alder = (3+2) * 3 + 18`

5 15 33

Videre om evaluering av uttrykk (forts.)

- For noen formål må man uansett ha paranteser
  - `alder = (3+2) * 3 + 18`

5 15 33

- Og for å gjøre det samme tydeligere - nøsting av paranteser er definitivt lovlig:
  - alder = ((3+2) * 3) + 18

Videre om evaluering av uttrykk (forts.)

- For noen formål må man uansett ha paranteser
  - alder = ~~(3+2)~~ ~~* 3~~ ~~+ 18~~ (5) (15) (33)
- Og for å gjøre det samme tydeligere - nøsting av paranteser er definitivt lovlig:
  - alder = ((3+2) * 3) + 18
- I praksis er det ikke tall man putter inn på slik måte, men variabler:
  - alder = ((bachelor+master) * antallFagfelt) + barndom

En liten oppgave om variable og uttrykk

(fra eksamen 2014)

Oppgave 1

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

```
int tall=(5+3)*2;
tall = tall+2;
```

Hvordan et program flyter fra linje til linje

- Dette er temmelig enkelt for det vi har lært frem til nå (endrer seg om en halvtime..)
- Hovedregel:
  - Gjør ferdig en linje, deretter gå til linjen nedenfor
- Ved en if-setning
  - Dersom det som testes er sant:
    gå til første linje i blokken
  - ellers:
    hopp over blokken

Hvordan et program flyter fra linje til linje

- Ved en if-else-setning
- Dersom det som testes er sant:
  gå til første linje i blokken etter if
- ellers:
  gå til første linje i blokken etter else

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

```
int lengde=7
int bredde=4
int omkrets

if lengde==bredde:
  omkrets = 4*lengde
 else:
  omkrets = (2*lengde) + (2*bredde)

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

Følge et program, linje for linje

```
int lengde=7
int bredde=4
int omkrets

if  7   ==bredde:
  omkrets = 4*lengde
 else:
  omkrets = (2*lengde) + (2*bredde)

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

Følge et program, linje for linje

```
int lengde=7
int bredde=4
int omkrets

if  7   ==   4 :
 omkrets = 4*lengde
 else:
  omkrets = (2*lengde) + (2*bredde)

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

Følge et program, linje for linje

```
int lengde=7
int bredde=4
int omkrets

if  7   ==   4 :
  omkrets = 4*lengde
 else:
  omkrets = (2* 7   ) + (2*bredde)

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

Følge et program, linje for linje

```
int lengde=7
int bredde=4
int omkrets

if  7   ==   4 :
 omkrets = 4*lengde
 else:
  omkrets = (   14   ) + (2*bredde)

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

Følge et program, linje for linje

```
int lengde=7
int bredde=4
int omkrets

if  7   ==   4 :
 omkrets = 4*lengde
 else:
  omkrets = (   14   ) + (2*   4   )

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

Følge et program, linje for linje

```
int lengde=7
int bredde=4
int omkrets

if  7   ==   4 :
 omkrets = 4*lengde
 else:
  omkrets = (   14   ) + (      8   )

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

Følge et program, linje for linje

```
int lengde=7
int bredde=4
int omkrets

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

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

Følge et program, linje for linje

```
int lengde=7
int bredde=4
int omkrets

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

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

Outline:

- Repetisjon fra forrige uke
- Lese inn fra tastaturet
- Beslutninger
- Kontrollflyt
- Prosedyrer

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

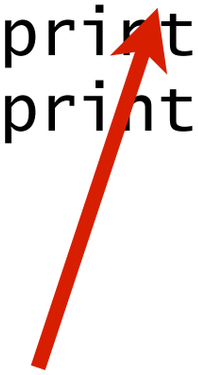
```
def giBeskjed():
  print("Jeg sier dette bare en gang!")
print("Da var jeg ferdig!")
```

Eksempel på prosedyre: kall

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



giBeskjed()
```

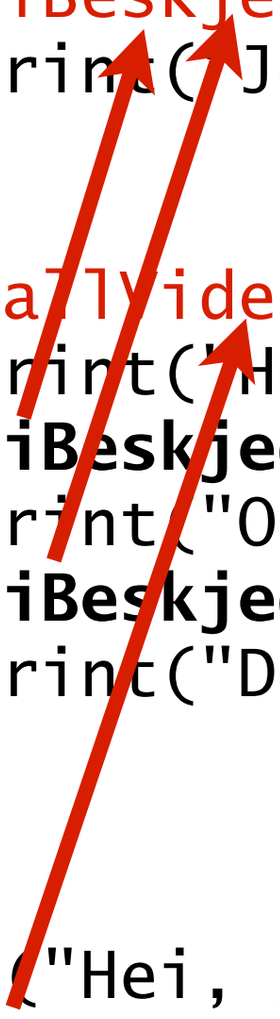


Kontrollflyt og metoder

```
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?")
```

Oppsummering

- Programmene blir ofte mer dynamiske når man leser inputt fra brukeren
- Beslutninger (if) lar programmene utføre ulike operasjoner avhengig av verdien til en variabel
- Programkode kjøres på en presist definert måte innad i en linje, og fra linje til linje
  - Å leke debugger med blyant og papir er nyttig!
- Subrutiner (prosedyrer) strukturerer kode og unngår redundans

Skriv (med blyant og papir)
en kode som bytter om verdiene mellom to variable

Prøv/tenk selv i 3 minutt.
Skriv bare linjene som trengs der det nå er rød kommentar.

```
tall1 = 4
tall2 = 9
#skriv koden som trengs her for å bytte om verdiene i
#variablene tall1 og tall2, uansett hva verdiene er
tall3 = tall1
tall1 = tall2
tall2 = tall3

assert tall1==9
assert tall2==4
print("Alt gikk bra!")
```