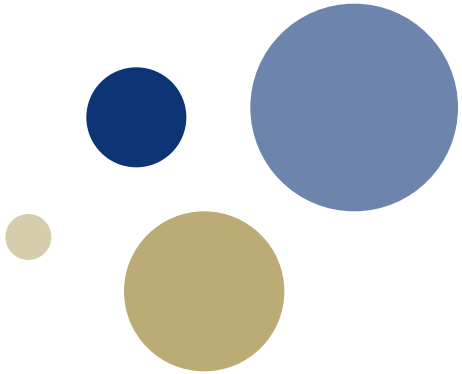
NTNU

Kunnskap for en bedre verden



Python: Variable og beregninger, innlesing fra tastatur utskrift til skjerm

TDT4110 IT Grunnkurs

Professor Guttorm Sindre

Python, pensum og ikke

Vi trenger å	Støttes av	
Hente data fra bruker	**•Fra tastatur: input()**	Andre former for input
Vise data til bruker	**•Tekst til skjerm: print() m.m.**	Andre former for output
Lagre data i minnet til bruk videre i programmet	**•Variable, enkle datatyper: int, float, bool, string**	**•…sammensatte datatyper: Lister, tupler, mengder, dictionary,** objekter / klasser
Lagre data permanent	**•Tekstfiler**	**•Binærfiler**
Prosessere data	**•Operatorer • = … +, -, … >, ==, …**	**•Innebygde funksjoner og metoder**
Styre hvorvidt og hvor ofte programsetninger utføres •Valg •Repetisjoner	**Kontrollstruktur •standard sekvens •if-setning •løkker (while, for)**	**•Unntaksbehandling •Rekursjon**
Gjøre programmet forståelig Bryte ned problemet i deler Oppnå fleksibilitet og gjenbrukbarhet	**•Kommentarer •Funksjoner •Moduler**	•Objektorientert design •Klasser og arv
Forstå hva vi har gjort feil	**•Feilmeldinger**	•Debugging

Denne uka

Vi trenger å	Støttes av	
Hente data fra bruker	•Fra tastatur: input()	Andre former for input
Vise data til bruker	•Tekst til skjerm: print() m.m.	Andre former for output
Lagre data i minnet for bruk videre i programmet	•Variable, enkle datatyper: Heltall, flyttall, strenger, sannhetsverdier	•…sammensatte datatyper: Lister, tupler, mengder, dictionary, objekter / klasser
Lagre data permanent (og hente)	•Tekstfiler	•Binærfiler
Prosessere data	•Operatorer • = , +=… +, -, *… >, ==, …	•Innebygde funksjoner og metoder
Styre hvorvidt og hvor ofte programsetninger utføres •Valg •Repetisjoner	Kontrollstruktur •standard sekvens •if-setning •løkker (while, for)	Kontrollstruktur •Unntaksbehandling •Rekursjon
Gjøre programmet forståelig Bryte ned problemet i deler Oppnå fleksibilitet og gjenbrukbarhet	•Kommentarer •Funksjoner •Moduler	•Objektorientert design •Klasser og arv
Forstå hva vi har gjort feil	•Feilmeldinger	•Debugging

Læringsmål og pensum

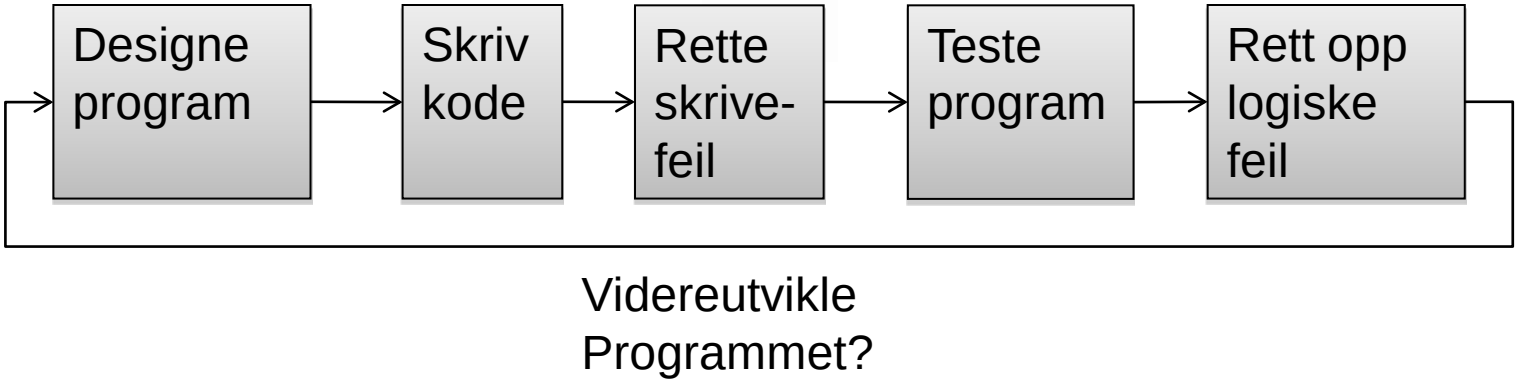
- Mål for denne uka:
  - Vite litt om design av programmer (2.1, 2.2, 2.4)
  - Lese inn data fra tastatur (2.6) og skrive til skjermen (2.3, 2.8)
  - Kunne bruke variable og vite om datatyper (2.5)
    - Konvertering mellom datatyper
    - ...og gjøre enkle beregninger (2.7)
- Pensum
  - […] Python Ch 2: Input, Processing, and Output

Designe et program

Kapittel 2.1-2.2

Programutviklingssyklus

- Programutviklingssyklusen er prosessen man følger når man lager og utvikler programmer:



Hjelp til å designe programmer: Pseudokode

- Beskriver programmet med naturlig språk
- Pseudokode kan ikke forstås av en datamaskin
  - men kan oversettes av mennesker til ulike programmeringsspråk

```
Hent inn (input) radius (r) for sylinderen
Hent inn (input) høyde (h) for sylinderen
Kalkuler areal og volum av sylinderen
    a) kalkuler omkrets av sirkel, O_sirkel = 2*pi*r
    b) areal av sirkel, A_sirkel = pi*r**2
    c) volum av sylinder,  A_sirkel*h
    d) areal av sylinder, 2*A_sirkel + omkrets*h
Vis (på skjerm) areal og volum
```

Nå

Vi trenger å	Støttes av	
Hente data fra bruker	•Fra tastatur: input()	Andre former for input
Vise data til bruker	•Tekst til skjerm: print() m.m.	Andre former for output
Lagre data i minnet for bruk videre i programmet	•Variable, enkle datatyper: Heltall, flyttall, strenger, sannhetsverdier	•…sammensatte datatyper: Lister, tupler, mengder, dictionary, objekter / klasser
Lagre data permanent (og hente)	•Tekstfiler	•Binærfiler
Prosessere data	•Operatorer • = , +=… +, -, *… >, ==, …	•Innebygde funksjoner og metoder
Styre hvorvidt og hvor ofte programsetninger utføres •Valg •Repetisjoner	Kontrollstruktur •standard sekvens •if-setning •løkker (while, for)	Kontrollstruktur •Unntaksbehandling •Rekursjon
Gjøre programmet forståelig Bryte ned problemet i deler Oppnå fleksibilitet og gjenbrukbarhet	•Kommentarer •Funksjoner •Moduler	•Objektorientert design •Klasser og arv
Forstå hva vi har gjort feil	•Feilmeldinger	•Debugging

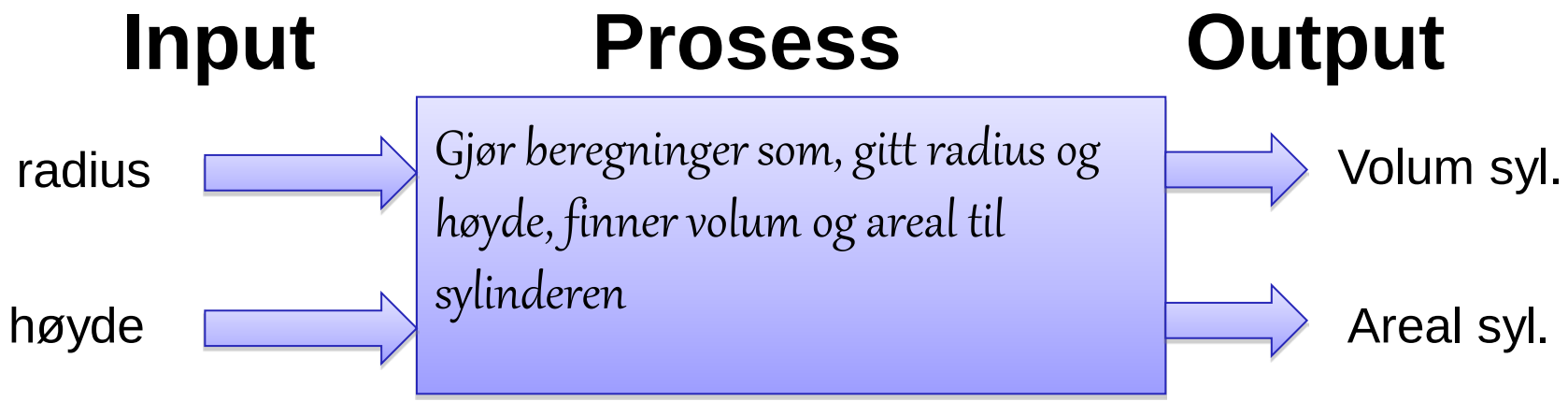
Kommentarer i programmer

- Forklarer hva som blir gjort i programmet
- Blir ignorert av tolker / kompilator
- Blir ikke skrevet ut til skjerm.
- Startes i Python med tegnet #

- Kommentarer er spesielt viktige
  - For store programmer
  - som flere samarbeider om
  - og som skal brukes lenge
- Men også ok i egen jobbing med små programmer:
  - Først skrive pseudokode som kommentarer
  - Så skrive programkoden mellom kommentarene

Hjelp til å designe programmer: Input – prosessering - output

- Dataprogrammer utføres ofte som en trestegsprosess:
  1. Mottar input
  2. Utfører prosessering av input (gjøre noe med input)
  3. Produserer output



Nå

Vi trenger å	Støttes av	
Hente data fra bruker	•Fra tastatur: input()	Andre former for input
Vise data til bruker	•Tekst til skjerm: print() m.m.	Andre former for output
Lagre data i minnet for bruk videre i programmet	•Variable, enkle datatyper: Heltall, flyttall, strenger, sannhetsverdier	•…sammensatte datatyper: Lister, tupler, mengder, dictionary, objekter / klasser
Lagre data permanent (og hente)	•Tekstfiler	•Binærfiler
Prosessere data	•Operatorer • = , +=… +, -, *… >, ==, …	•Innebygde funksjoner og metoder
Styre hvorvidt og hvor ofte programsetninger utføres •Valg •Repetisjoner	Kontrollstruktur •standard sekvens •if-setning •løkker (while, for)	Kontrollstruktur •Unntaksbehandling •Rekursjon
Gjøre programmet forståelig Bryte ned problemet i deler Oppnå fleksibilitet og gjenbrukbarhet	•Kommentarer •Funksjoner •Moduler	•Objektorientert design •Klasser og arv
Forstå hva vi har gjort feil	•Feilmeldinger	•Debugging

Eksempel

- Vi ønsker å lage et program med følgende oppførsel
  - ...der blå tekst skrives av maskinen, svart tekst av brukeren

```
============== RESTART: /Users/guttorm/Documents/smartere.
Hva er navnet ditt? Guttorm
Guttorm, beskriv deg selv med et adjektiv: smart
Hah! Så du tror du er smart?
Jeg, Maskinen, er mye smartere...
>>>
```

1. Få brukeren til å oppgi navnet sitt via tastaturet
2. Få brukeren til å beskrive seg selv med et adjektiv
   - Her skal navn fra pkt 1 med i ledeteksten (mer ”personlig”)
3. Maskinen disser brukeren, sier den selv er bedre
   - Legge til ’ere’ på oppgitt adjektiv

To funksjoner vi trenger her

- input() - lar brukeren skrive tekst via tastaturet
  - Vilkårlig antall tegn, avsluttes med ENTER-tasten
  - Kan ta null eller ett argument, hvis ett brukes dette som ledetekst
- print() - viser tekst på skjermen
  - Kan ta null, ett eller flere (vilkårlig mange) argumenter
  - som kan være tekster, tall, og andre typer data
  - Avslutter med linjeskift (hvis ikke annet er spesifikt oppgitt)

- Mhp antall argumenter ~ norsk grammatikk
  - Noen verb er intransitive (null objekt): *"Bilen eksploderte."*
  - Noen er transitive (ett objekt): *"Superhelten sprengte bilen."*
  - …(to objekter, indirekte og direkte): *"Jeg tilbød Carlsen remis."*
  - …ulike antall: *"Kan du gi?", "Gi jernet!", "Gi meg gaven!"*

Første (dårlig) forsøk på løsning

```
# Lese inn navn fra tastaturet
input('Hva er navnet ditt? ')

# Lese inn et adjektiv fra tastaturet
input('Guttorm, beskriv deg selv med et adjektiv: ')

# Disse brukeren, si Maskinen er bedre
print('Hah! Så du tror du er smart?')
print('Jeg, Maskinen, er mye smartere...')
```

- Gjør det konkrete eksemplet tilsynelatende riktig...
- MEN: skriver samme navn (Guttorm) og samme adjektiv (smart) uansett hva brukeren skrev
- For å fikse dette, må vi HUSKE dataene brukeren skriver inn
  - nå glemmes de umiddelbart
- Trenger VARIABLE for å huske data underveis i et program

Bedre løsning

```
# Lese inn navn fra tastaturet, HUSK det i variabelen navn
navn = input('Hva er navnet ditt? ')

# Lese inn et adjektiv fra tastaturet, HUSK det i variabelen adj
adj = input(navn + ', beskriv deg selv med et adjektiv: ')

# Disse brukeren, si Maskinen er bedre
print('Hah! Så du tror du er ' + adj + 'L?')
print('Jeg, Maskinen, er mye ' + adj + 'ere...')
```

Bedre løsning


```
# Lese inn navn fra tastaturet, HUSK det i variabelen navn
navn = input('Hva er navnet ditt? ')

# Lese inn et adjektiv fra tastaturet, HUSK det i variabelen adj
adj = input(navn + ', beskriv deg selv med et adjektiv: ')

# Disse brukeren, si Maskinen er bedre
print('Hah! Så du tror du er ' + adj + 'L?')
print('Jeg, Maskinen, er mye ' + adj + 'ere...')
```

Nå

Vi trenger å	Støttes av	
Hente data fra bruker	•Fra tastatur: input()	Andre former for input
Vise data til bruker	•Tekst til skjerm: print() m.m.	Andre former for output
Lagre data i minnet for bruk videre i programmet	•Variable, enkle datatyper: Heltall, flyttall, strenger, sannhetsverdier	•…sammensatte datatyper: Lister, tupler, mengder, dictionary, objekter / klasser
Lagre data permanent (og hente)	•Tekstfiler	•Binærfiler
Prosessere data	•Operatorer • = , +=… +, -, *… >, ==, …	•Innebygde funksjoner og metoder
Styre hvorvidt og hvor ofte programsetninger utføres •Valg •Repetisjoner	Kontrollstruktur •standard sekvens •if-setning •løkker (while, for)	Kontrollstruktur •Unntaksbehandling •Rekursjon
Gjøre programmet forståelig Bryte ned problemet i deler Oppnå fleksibilitet og gjenbrukbarhet	•Kommentarer •Funksjoner •Moduler	•Objektorientert design •Klasser og arv
Forstå hva vi har gjort feil	•Feilmeldinger	•Debugging

Hvorfor trenger vi variable?

- Forrige eksempel: trenger å huske data som brukeren skriver inn via tastatur
- Også huske andre data i programmet
- F.eks. konstanter, resultat av mellomregninger
- Eksempel (dårlig løsning):

```
pi = 3.14
r = 5.2
h = 7.9
print('Omkrets av sirkelen er', 2 * pi * r)
print('Areal av sirkelen er', pi * r ** 2)
print('Volum av sylinderen er', pi * r ** 2 * h)
print('Areal av sylinderen er', 2 * pi * r ** 2 + 2 * pi * r * h)
```

Hvorfor trenger vi variable?

- Forrige eksempel: trenger å huske data som brukeren skriver inn via tastatur
- Også huske andre data trengs i programmet
- F.eks. konstanter, resultat av mellomregninger
- Eksempel (dårlig løsning):

```
pi = 3.14
r = 5.2
h = 7.9
print('Omkrets av sirkelen er', 2 * pi * r)
print('Areal av sirkelen er', pi * r ** 2)
print('Volum av sylinderen er', pi * r ** 2 * h)
print('Areal av sylinderen er', 2 * pi * r ** 2 + 2 * pi * r * h)
```

Regner ut på nytt omkrets og areal enda vi har regnet det før
Sløser tid og strøm!

Bedre løsning

```
pi = 3.14
r = 5.2
h = 7.9
omkrets_sirkel = 2 * pi * r
print('Omkrets av sirkelen er', omkrets_sirkel)
areal_sirkel = pi * r ** 2
print('Areal av sirkelen er', areal_sirkel)
volum_sylinder = areal_sirkel * h
print('Volum av sylinderen er', volum_sylinder)
areal_sylinder = 2 * areal_sirkel + omkrets_sirkel * h
print('Areal av sylinderen er', areal_sylinder)
```

Bedre løsning

```
pi = 3.14
r = 5.2
h = 7.9
omkrets_sirkel = 2 * pi * r
print('Omkrets av sirkelen er', omkrets_sirkel)
areal_sirkel = pi * r ** 2
print('Areal av sirkelen er', areal_sirkel)
volum_sylinder = areal_sirkel * h
print('Volum av sylinderen er', volum_sylinder)
areal_sylinder = 2 * areal_sirkel + omkrets_sirkel * h
print('Areal av sylinderen er', areal_sylinder)
```

Variable

- Formål: huske data til bruk senere i programmet
  - Variabelen har et navn
  - Representerer en verdi lagret i datamaskinens minne
  - Innhold kan variere i løpet av programmet
- Oppretting av variable gjøres ved tilordningsuttrykk

```
variabelnavn = uttrykk
```

- = er ikke matematisk likhet men tilordning
- F.eks.

```
fornavn = 'Lise'
temperatur = 22.3
areal_sirkel = 2 * 3.14 * 5.6
fullt_navn = fornavn + ' Olsenius'
cirka_areal = round(areal_sirkel, 1)
```

Regler for tilordningssetninger

- Venstre side MÅ være et variabelnavn
- Høyre side MÅ være et uttrykk
  - enkelt eller sammensatt
  - som kan regnes ut til en verdi

```
>>> x = 1
>>> x = x + 1
>>> pris_ekskl_moms = 3000
>>> pris_inkl_moms = pris_ekskl_moms * 1.25
```

- Resultat når en tilordningssetning kjøres
  1. Verdien av uttrykket på høyre side regnes ut
  2. Det settes av plass til variabelen i minnet
  3. Verdien av uttrykket huskes i variabelen med gitt navn
- En variabel MÅ være opprettet før den kan brukes

Regler for variabelnavn

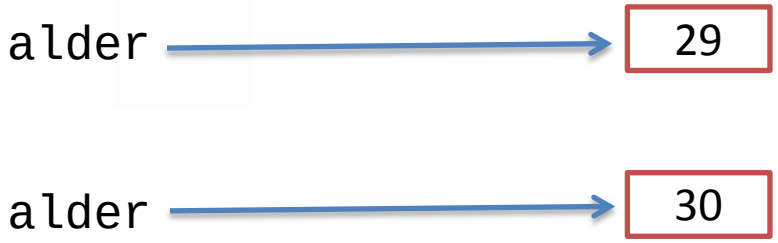
- Forbudt å bruke
  - Nøkkelord i Python (f.eks. def, if, …) som variabelnavn
  - Andre symboler enn bokstav eller _ som første tegn
  - Andre symboler enn bokstav, tall eller _ i påfølgende tegn

- Anbefalinger
  - Bruk navn som er informative, ikke villedende eller intetsigende
  - Unngå variabelnavn som lett kan forveksles / feiltastes
    - Python skiller mellom store og små bokstaver
      - Pris og pris vil være forskjellige variable, men ikke lurt å benytte dette
  - Hvis variabelnavn består av flere ord, bruk _ mellom ordene
    - F.eks. areal_sirkel, pris_inkl_moms
    - Eller stor bokstav på nytt ord: arealSirkel, prisInklMoms

Endring av verdi i en variabel

- En variabel kan endre verdi i løpet av et program.
- Man endrer verdien av variabel ved å gjøre en ny tilordning.
- Eks:

```
alder = 29
print(alder)
alder = 30
print(alder)
```



Lagring av verdier

- Variablenes verdier lagres sammen med andre data og programmer i minnet (RAM på datamaskinen)
  - Tenk på RAM som en kommode med nummererte skuffer
  - I hver skuff kan du lagre en *byte* eller 8 *biter* (bits)
  - En byte er et tall i området 0-255 (8 sifre i 2-tallssystemet)
- Pythons tolker / kompilator finner selv hvor mye plass som trengs for å lagre en verdi
  - En enkelt byte: lite heltall eller bokstav
  - Flere bytes: store heltall, flyttall
  - Mange bytes: lange tekststrenger, lister, mengder

Nå

Vi trenger å	Støttes av	
Hente data fra bruker	•Fra tastatur: input()	Andre former for input
Vise data til bruker	•Tekst til skjerm: print() m.m.	Andre former for output
Lagre data i minnet for bruk videre i programmet	•Variable, enkle datatyper: Heltall, flyttall, strenger, sannhetsverdier	•…sammensatte datatyper: Lister, tupler, mengder, dictionary, objekter / klasser
Lagre data permanent (og hente)	•Tekstfiler	•Binærfiler
Prosessere data	•Operatorer • = , +=… +, -, *… >, ==, …	•Innebygde funksjoner og metoder
Styre hvorvidt og hvor ofte programsetninger utføres •Valg •Repetisjoner	Kontrollstruktur •standard sekvens •if-setning •løkker (while, for)	Kontrollstruktur •Unntaksbehandling •Rekursjon
Gjøre programmet forståelig Bryte ned problemet i deler Oppnå fleksibilitet og gjenbrukbarhet	•Kommentarer •Funksjoner •Moduler	•Objektorientert design •Klasser og arv
Forstå hva vi har gjort feil	•Feilmeldinger	•Debugging

Datatyper

- I Python kan variable ta vare på ulike typer data (datatyper)
- Nå elementære datatyper (en verdi per variabel):
  - Heltall (int): antall = 7
  - Desimaltall (float): penger = 35.5 # Bruker . i stedet for ,
  - Tekststreng (str): navn = 'Petter' # Bruker fnutter
  - Sannhetsverdi (bool): rykte = True eller rykte = False
- Senere på høsten: sammensatte datatyper, f.eks. mengder
  - nordiske_land = { 'Norge', 'Sverige', 'Danmark', 'Finland', 'Island' }
- Hvilken datatype et uttrykk har kan sjekkes med

  type(uttrykk)
- I Python får variable type ved tilordning
  - Et variabelnavn kan bytte datatype underveis i programmet

Oppgave: Datatyper

- Angi hvilken datatype disse variablene bør være av heltall (integer), desimaltall (float), streng (string), sannhetsverdi (boolean):
  - A) Variabel for å lagre et telefonnummer
  - B) Variabel for å lagre millimeter nedbør
  - C) Variabel for å lagre navn på et fag
  - D) Variabel for å lagre om en deltaker er påmeldt eller ikke
  - E) Variabel for å lagre pris på bensin
  - F) Variabel for å lagre personnummer
  - G) Variabel for å lagre en tekstmelding

Konvertering mellom datatyper

- Vanlig problem: vi har data av ”feil” type
- Trenger å konvertere til ønsket datatype
- Nyttige funksjoner
  - int() konverterer til heltall
    - Eks: int(3.75) gir 3 ; int(’42’) gir 42
  - float() konverterer til flyttall
    - Eks: float(3) gir 3.0 ; float(’3.75’) gir 3.75
  - str() konverterer til tekststreng
    - Eks: str(42) gir ’42’ ; str(3.75) gir ’3.75’
  - bool() konverterer til boolsk verdi (sannhetsverdi)
    - gir False for bool(0), bool(0.0), bool(’’)
    - gir True for alle tall ulik null og alle tekster annet enn tom streng
      - NB: gir også True for bool(’False’) og bool(’ ’) (space)

Bruk av int() og float()

- To måter å hente heltall fra input-funksjon:

```
streng_verdi = input('Hvor mange kuer har du ? ')
kuer = int(streng_verdi)          # Oversetter streng til heltall
```

- Du kan gjøre begge deler i en setning:

```
kuer = int(input('Hvor mange kuer har du ? '))
```

  –Kalles nøstet funksjonskall og fungerer på følgende måte:
  - Først utføres den innerste funksjonen, altså input('Hvor mange..')
  - Verdien til input-funksjonen brukes så i int-funksjonen
  - Resultatet fra int-funksjonen lagres i variabelen kuer

- Hente ut flyttall:

```
pi_tips = float(input("Gjett en verdi for tallet pi? "))
```

Eksempel: Behov for konvertering

- input()-funksjonen returnerer alltid en tekststreng
  - Dvs. en tekst med fnutter rundt.
  - Tekst funker dårlig i matematiske beregninger
- Vi ønsker et program med følgende oppførsel

```
>>> ================================ RESTART ==============
>>>
Din alder (antall år)? 52
Din kjærestes alder? 46
Aldersforskjellen er 6 år, stemmer det? Ja
Du svarte:  Ja
>>>
```

- Vedlagte aldersforskjell_V0 funker ikke
- OPPGAVE: Prøv å fikse den så den virker

Nå

Vi trenger å	Støttes av	
Hente data fra bruker	•Fra tastatur: input()	Andre former for input
Vise data til bruker	•Tekst til skjerm: print() m.m.	Andre former for output
Lagre data i minnet for bruk videre i programmet	•Variable, enkle datatyper: Heltall, flyttall, strenger, sannhetsverdier	•…sammensatte datatyper: Lister, tupler, mengder, dictionary, objekter / klasser
Lagre data permanent (og hente)	•Tekstfiler	•Binærfiler
Prosessere data	•Operatorer • = , +=… +, -, *… >, ==, …	•Innebygde funksjoner og metoder
Styre hvorvidt og hvor ofte programsetninger utføres •Valg •Repetisjoner	Kontrollstruktur •standard sekvens •if-setning •løkker (while, for)	Kontrollstruktur •Unntaksbehandling •Rekursjon
Gjøre programmet forståelig Bryte ned problemet i deler Oppnå fleksibilitet og gjenbrukbarhet	•Kommentarer •Funksjoner •Moduler	•Objektorientert design •Klasser og arv
Forstå hva vi har gjort feil	•Feilmeldinger	•Debugging

Operatorpresedens
(prioritert rekkefølge)

- Python har følgende matteoperatorer: + - * / // % **
- Rekkefølge av utregning avhenger av presedens:
  - Potens: **
  - Minus som fortegn: -
  - Multiplikasjon, divisjon, heltallsdivisjon og modulo: * / // %
  - Addisjon og subtraksjon: + −
  - …
  - Tilordning: =
  - Lik presedens? Da utregnes det fra venstre mot høyre
- For å sikre korrekte beregninger brukes parenteser:
  - (12+6)/3 blir 6
  - 10/(5-3) blir 5 osv…
  - Nøstede parenteser løses innenfra og ut

Hva funker operatorene for?

+	int	float	str	bool
int	int	float		int
float	float	float		float
str			str	
bool	int	float		int

-	int	float	str	bool
int	int	float		int
float	float	float		float
str				
bool	int	float		int

*	int	float	str	bool
int	int	float	str	int
float	float	float		float
str	str			str
bool	int	float	str	int

/	int	float	str	bool
int	float	float		float
float	float	float		float
str				
bool	float	float		float

//	int	float	str	bool
int	int	float		int
float	float	float		float
str				
bool	int	float		int

%	int	float	str	bool
int	int	float		int
float	float	float		float
str				
bool	int	float		int

FEIL

OK

Funker, men rart

Programmere litt?

- Oppgave: Lag et program som
  - Henter inn tre måleverdier
  - Kalkulerer gjennomsnittet (summen delt på 3)
  - Viser resultat på skjerm
  - Eksempel på kjøring:

```
>>> ================================ RESTART ================================
>>>
Skriv inn måling 1: 2.43
Skriv inn måling 2: 5.67
Skriv inn måling 3: 6.2224
Måling 1: 2.43 Måling 2: 5.67 Måling 3: 6.2224 Snitt: 4.774133333333333
>>>
```

- Prøv å programmere dette selv!
  - Hvis du er så god at det blir enkelt: Lag penere utskrift
    - Bruk tabulator eller format
    - Vis at du ville klare å få tall i pene kolonner under hverandre hvis det skulle regnes ut flere gjennomsnitt

Nå

Vi trenger å	Støttes av	
Hente data fra bruker	•Fra tastatur: input()	Andre former for input
Vise data til bruker	•Tekst til skjerm: print() m.m.	Andre former for output
Lagre data i minnet for bruk videre i programmet	•Variable, enkle datatyper: Heltall, flyttall, strenger, sannhetsverdier	•…sammensatte datatyper: Lister, tupler, mengder, dictionary, objekter / klasser
Lagre data permanent (og hente)	•Tekstfiler	•Binærfiler
Prosessere data	•Operatorer • = , +=… +, -, *… >, ==, …	•Innebygde funksjoner og metoder
Styre hvorvidt og hvor ofte programsetninger utføres •Valg •Repetisjoner	Kontrollstruktur •standard sekvens •if-setning •løkker (while, for)	Kontrollstruktur •Unntaksbehandling •Rekursjon
Gjøre programmet forståelig Bryte ned problemet i deler Oppnå fleksibilitet og gjenbrukbarhet	•Kommentarer •Funksjoner •Moduler	•Objektorientert design •Klasser og arv
Forstå hva vi har gjort feil	•Feilmeldinger	•Debugging

Strenger omsluttes på flere måter

- Enkeltfnutter ’…..’
- Dobbelfnutter ”…..”
- Triple fnutter (enkle eller doble) ’’’…..’’’ eller ”””….”””
- Smak og behag, men noen ganger funker det ene ikke
  - print(”Don’t laugh”)
  - print(’Da ”Joe” red inn i byen ved daggry…’)
  - print(’’’Don’t think ”Joe” loves you’’’)

- Viktig å skjønne at det er forskjell på strenger og variabelnavn / uttrykk

Escape-karakterer

- Escape-karakterer er spesialtegn
  - kan skrives inn i tekststrenger for spesielle operasjoner:

\n Gir linjeskift

\t Hopper til neste tabulator

\’ Skriver ut tegnet ’

\” Skriver ut tegnet ”

\\ Skriver ut tegnet \

–Eks:

```
print('\tInnrykk er kult med\nNy linje')
```

Endring av oppførsel for print()

- Endre separasjon mellom elementer

```
a=5
b=7
print(a, b)                 # gir 5  7
print(a, b, sep='_')        # gir 5_7
print(a, b, sep='')         # gir 57
```

- Annen avslutning enn linjeskift

```
print('Her kommer en setning.', end='')
print('her kommer en setning til.')
```

  - Begge blir skrevet ut på samme linje!
  - For dobbelt linjeskift etter print: end = '\n\n'

Pen utskrift i kolonner

- For tall: Funksjonen format
  - tar inn et tall og en tekststreng som viser format
  - Returnerer tekststreng hvor tallet er konvertert til dette formatet

  format(tall, formattering)

  –formattering er en tekststreng for ulike valg, f.eks:

```
format(1/3, '.2f')      # 2 desimaler, f står for float
format(1/3, '10.1f')    # 1 desimal og setter av 10 tegn
format(1/3, 'e')        # vitenskapelig notasjon på tallet
format(1/3, '.0%')      # tallet i prosent med 0 desimaler
format(500, '10d')      # heltall der det settes av 10 tegn
```

  –Typisk bruk:

```
print(format(1/3,'10.5f'))
```

- For strenger: mulig å justere til venstre og høyre med
  - –tekst.ljust() og tekst.rjust() – angi bredde i parentesen

Oppsummering

- Utviklingssyklus:
  - Design, programmer, rett opp skrivefeil, test, rett logiske feil
- Skrive til skjerm: print(uttrykk)
  - Kan ha flere uttrykk etter hverandre
  - Formattering (pen utskrift): format(tall, 'formatstreng')
- Variabler: Referanser med navn til verdier i minnet
- Tilordning: Gi variable verdier: **variabel = uttrykk**
  - Evt. variable på høyre side av **=** bidrar med verdier i regnestykket
  - Resultatet blir lagret i variabel på venstre side
- Basis datatyper: int, float, str, bool
  - Bruk av feil type gir ofte kluss
- Lese fra tastatur: variabel = input(prompt)
- Operatorpresedens: Rekkefølge av matematiske operatorer
- Formattering av tall ved hjelp av funksjonen format()