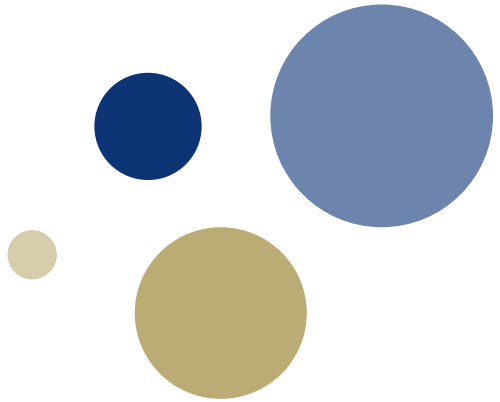
NTNU

Kunnskap for en bedre verden



Python: Intro til funksjoner

TDT4110 IT Grunnkurs

Professor Guttorm Sindre

Snart referansegruppemøte

- Viktig mulighet for å gi tilbakemelding på emnet
  - Pensumbøker
  - Forelesninger
  - Øvingsforelesninger
  - Veiledning på sal
  - Øvingsoppgaver
  - Andre læringsressurser (kollokvier, Piazza, videoer…)
- Ros, kritikk, forbedringsforslag…
- Jeg går ut av auditoriet ca. 10 min før pause i dag
  - Så kan referansegruppemedl. fasilitere en diskusjon
  - Også mulig å kontakte ref.gr.medl. på andre måter
    - Epost, pauser, …
  - Se itgk.idi.ntnu.no for hvem som er i ref.gr. fra din klasse

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
  - Forstå hvorfor programmer deles i funksjoner
  - Bli i stand til å definere og kalle funksjoner
  - Klare å bruke lokale variable
  - Klare å overføre parametre til funksjoner
  - Lære fornuftig bruk av globale variable og konstanter
- Pensum
  - 3.utg. …Python: Ch. 5.1-5.7 (2.utg. Ch. 3)

Introduksjon til funksjoner

Kapittel 5.1

Funksjoner: Hva og hvorfor?

- **Funksjon:**
  - en gruppe kodelinjer som utfører en spesifikk oppgave
  - kan minne om matematiske funk., men Python-funksjoner…
    - Kan inneholde alle slags handlinger, ikke bare matematikk
    - Kan returnere et svar (som mat.funk.), men kan også la være
    - Kan ta argumenter, men kan også la være
- Hvorfor bruke funksjoner?
  - Dele et større program opp i mindre deler
    - Lettere å forstå, enklere å teste, mer effektivt samarbeid
  - Gjenbruk av kode
    - Vi kan bruke funksjoner andre har skrevet, f.eks. print(), round()
    - Vi kan selv definere nye funksjoner og
      - bruke dem flere sted i ett program,
      - eller i flere ulike programmer

Eksempel

- Skal beregne og presentere forskningsdata
- Vil ha en tydelig innrammet overskrift foran hver tabell med data

```
****************************************
* HER ER DATAENE                       *
****************************************
```

- Forskningsdataene beregnes på ulike måter, men skal vises med samme slags overskrift mange sted i programmet. Irriterende å gjenta koden mange sted:

```
print('*' * 40)
print('*' + ' ' + 'HER ER DATAENE' + ' ' * 23 + '*')
print('*' * 40)
```

- Definerer det i stedet som en funksjon

```
def print_data_heading():
    print('*' * 40)
    print('*' + ' ' + 'HER ER DATAENE' + ' ' * 23 + '*')
    print('*' * 40)
```

- Får da utført dette med bare en kodelinje flere sted i programmet:

```
print_data_heading() # kaller funksjonen, som dermed blir utført
```

Eksempel (forts.)

```
def print_data_heading():
    print('*' * 40)
    print('*' + ' ' + 'HER ER DATAENE' + ' ' * 23 + '*')
    print('*' * 40)

print('Eksperimentet ga følgende resultat:')
print_data_heading() # kaller funksjonen, som dermed blir utført
for i in range(25):  # lager noen tulletall for syns skyld
    print(format(9.5-i**(0.5),'6.2f'), end=' '+'\n'*(i%5==4))
print()
print()

print('Rådataene fra spørreundersøkelsen er som følger:')
print_data_heading() # kaller funksjonen, som dermed blir utført
for i in range(24):  # lager noen tulletall for syns skyld
    print(format(9-int(i**(0.5)),'4d'), end=' '+'\n'*(i%6==5))
print()
print()
```

Funksjonen print_data_heading() defineres her

Kan så brukes flere sted i koden

(kunne vært enda flere)

KODE: headings_V1.py

headings_V0.py viser UTEN funksjon, må da gjenta lik kode

Hvordan kjøres dette programmet?

Registrerer at det defineres en funksjon, Utfører ikke linjene inni nå, bare husker def. til senere

```
def print_data_heading():
    print('*' * 40)
    print('*' + ' ' + 'HER ER DATAENE' + ' ' * 23 + '*')
    print('*' * 40)

print('Eksperimentet ga følgende resultat:')
print_data_heading() # kaller funksjonen, som dermed blir utført
for i in range(25):  # lager noen tulletall for syns skyld
    print(format(9.5-i**(0.5),'6.2f'), end=' '+'\n'*(i%5==4))
print()
print()

print('Rådataene fra spørreundersøkelsen er som følger:')
print_data_heading() # kaller funksjonen, som dermed blir utført
for i in range(24):  # lager noen tulletall for syns skyld
    print(format(9-int(i**(0.5)),'4d'), end=' '+'\n'*(i%6==5))
print()
print()
```

Hvordan kjøres dette programmet?

Fortsetter etter definisjonen, ”vanlig” kodelinje som utføres som normalt

```
def print_data_heading():
    print('*' * 40)
    print('*' + ' ' + 'HER ER DATAENE' + ' ' * 23 + '*')
    print('*' * 40)

print('Eksperimentet ga følgende resultat:')
print_data_heading() # kaller funksjonen, som dermed blir utført
for i in range(25):  # lager noen tulletall for syns skyld
    print(format(9.5-i**(0.5),'6.2f'), end=' '+'\n'*(i%5==4))
print()
print()

print('Rådataene fra spørreundersøkelsen er som følger:')
print_data_heading() # kaller funksjonen, som dermed blir utført
for i in range(24):  # lager noen tulletall for syns skyld
    print(format(9-int(i**(0.5)),'4d'), end=' '+'\n'*(i%6==5))
print()
print()
```

Hvordan kjøres dette programmet?

Neste kodelinje er et kall til den definerte funksjonen. Da hopper utførelsen dit…

```
def print_data_heading():
    print('*' * 40)
    print('*' + ' ' + 'HER ER DATAENE' + ' ' * 23 + '*')
    print('*' * 40)

print('Eksperimentet ga følgende resultat:')
print_data_heading() # kaller funksjonen, som dermed blir utført
for i in range(25):  # lager noen tulletall for syns skyld
    print(format(9.5-i**(0.5),'6.2f'), end=' '+'\n'*(i%5==4))
print()
print()

print('Rådataene fra spørreundersøkelsen er som følger:')
print_data_heading() # kaller funksjonen, som dermed blir utført
for i in range(24):  # lager noen tulletall for syns skyld
    print(format(9-int(i**(0.5)),'4d'), end=' '+'\n'*(i%6==5))
print()
print()
```

Hvordan kjøres dette programmet?

Og utfører i rekkefølge de tre kode-linjene i funksjonen

utførelsen dit…

```
def print_data_heading():
    print('*' * 40)
    print('*' + ' ' + 'HER ER DATAENE' + ' ' * 23 + '*')
    print('*' * 40)

print('Eksperimentet ga følgende resultat:')
print_data_heading() # kaller funksjonen, som dermed blir utført
for i in range(25):  # lager noen tulletall for syns skyld
    print(format(9.5-i**(0.5),'6.2f'), end=' '+'\n'*(i%5==4))
print()
print()

print('Rådataene fra spørreundersøkelsen er som følger:')
print_data_heading() # kaller funksjonen, som dermed blir utført
for i in range(24):  # lager noen tulletall for syns skyld
    print(format(9-int(i**(0.5)),'4d'), end=' '+'\n'*(i%6==5))
print()
print()
```

Hvordan kjøres dette programmet?

Etter at tredje linje inni funksjonen er ferdig, vender utførelsen tilbake dit kallet kom fra. Fortsetter med neste linje, dette er ei løkke som kjører 25 ganger…

```
def print_data_heading():
    print('*' * 40)
    print('*' + ' ' * 12 + 'HER ER DATAENE' + ' ' * 12 + '*')
    print('*' * 40)

print('Eksperimentet ga følgende resultat:')
print_data_heading()
for i in range(25): #lager bare noen tulle-tall for syns skyld
    print(format(9.5-i**(0.5),'6.2f'), end=' '+'\n'*(i%5==4))
print()
print()

print('Rådataene fra spørreundersøkelsen er som følger:')
print_data_heading()
for i in range(24): #lager bare noen tulle-tall for syns skyld
    print(format(9-int(i**(0.5)),'4d'), end=' '+'\n'*(i%6==5))
print()
print()
```

Hvordan kjøres dette programmet?

```
def print_data_heading():
    print('*' * 40)
    print('*' + ' ' * 12 + 'HER ER DATAENE' + ' ' * 12 + '*')
    print('*' * 40)

print('Eksperimentet ga følgende resultat:')
print_data_heading()
for i in range(25): #lager bare noen tulle-tall for syns skyld
    print(format(9.5-i**(0.5),'6.2f'), end=' '+'\n'*(i%5==4))
print()
print()

print('Rådataene fra spørreundersøkelsen er som følger:')
print_data_heading()
for i in range(24): #lager bare noen tulle-tall for syns skyld
    print(format(9-int(i**(0.5)),'4d'), end=' '+'\n'*(i%6==5))
print()
print()
```

Når vi etter hvert kommer til neste funksjonskall, hopper utførelsen igjen til funksjonens kode

Hvordan kjøres dette programmet?

```
def print_data_heading():
    print('*' * 40)
    print('*' + ' ' * 12 + 'HER ER DATAENE' + ' ' * 12 + '*')
    print('*' * 40)

print('Eksperimentet ga følgende resultat:')
print_data_heading()
for i in range(25): #lager bare noen tulle-tall for syns skyld
    print(format(9.5-i**(0.5),'6.2f'), end=' '+'\n'*(i%5==4))
print()
print()

print('Rådataene fra spørreundersøkelsen er som følger:')
print_data_heading()
for i in range(24): #lager bare noen tulle-tall for syns skyld
    print(format(9-int(i**(0.5)),'4d'), end=' '+'\n'*(i%6==5))
print()
print()
```

…og tilbake igjen der vi slapp i hovedskriptet når funksjonen er ferdig

Definere og kalle funksjoner

Kapittel 5.2

Lage en funksjon

- Generelt oppsett for definisjon av funksjon:

```
def funksjons_navn():
    kode
    kode
    etc.
```

–Første linje kalles funksjonshode:

- Markerer starten på funksjon med det reserverte ordet def,
- fulgt av navnet på funksjonen, parenteser og et kolon

–Resten av koden kalles kodeblokk som hører til funksjonen

- **NB! Denne må skrives med innrykk!!!**
- (samme som for if, while, for… viser hvilken kode som hører til)

Bruk av flere funksjoner

- Programmer kan bygges av flere funksjoner.
- Vanlig praksis: hovedfunksjon kalles main( ):
  - hovedlogikken i programmet
  - gjengir overordnet struktur i programmet
  - kaller andre funksjoner som er definert

- Men vi må ha minst en kodelinje utenfor main()
  - ”skriptet” / ”hovedprogrammet”
  - Minimal løsning: her utføres kun et kall av main()

Overføring av argumenter til funksjoner

Kapittel 5.5

Argumenter og parametre

- Funksjon uten parametre, f.eks. def print_data_heading( ):
  - Lite fleksibelt, eksakt samme overskrift hver gang

```
****************************************
* HER ER DATAENE                       *
****************************************
```

  - Med parameter(e): ulike overskrifter fra gang til gang
- Parameter:
  - Står inni funksjonsparentesen i def-linja
  - Fungerer som en lokal variabel i funksjonskoden
- Argument:
  - Står inni funksjonsparentesen der den kalles
  - Kan være en verdi, variabel, eller uttrykk som blir en verdi
  - Argumentverdien overføres til funksjonen,
    - Dvs. puttes inn for parametervariabelen når funksjonen utføres

Argumenter -> parametre

- Vanligst:
  - Samme antall argumenter som parametre
  - Samme rekkefølge
    - Kalles ”positional arguments”
    - Se kodeeksempel headings_V2.py, headings_V3.py
- Korrekt rekkefølge kan avvikes i kallet hvis vi
  - oppgir hvilket argument som skal gis til hvilken parameter
    - Kalles ”keyword arguments”
    - Se kodeeksempel headings_V3.py (siste linje i koden)
- Kan også i noen tilfeller avvike antall
  - Bruke færre argumenter enn funksjonen har parametre
    - NB: Kun hvis de som droppes har default-verdier
    - Se kodeeksempel headings_V4.py

Lokale og globale variable

Kapittel 5.4 + 5.6

Lokale vs. globale variable

- En lokal variabel
  - blir opprettet inni en funksjon
  - skal kun brukes lokalt i funksjonen
  - er umulig å nevne i kode utenom variabelens skop
- En global variabel
  - Defineres i hovedprogrammet
  - Er synlig der og i alle programmets funksjoner
  - Bruk av globale variable
    - Kan uten videre brukes / vises av alle funksjoner
    - Kan endres hvis man i funk. skriver **global variabelnavn**
  - IKKE anbefalt, annet enn for konstanter

Se eksempel fylkesmann_u_forklaring.py / fylkesmann_m_forklaring.py

Oppgave

Start med programmet **tabell_v0.py**. Lag en funksjon som kan printe ut en tabell som vist i eksempel på kjøring

```
Hvor mange rader ønsker du i tabellen? 4
Hvor mange kolonner ønsker du? 3
Kolonnebredde (antall blanke)? 3
-------------
|   |   |   |
-------------
|   |   |   |
-------------
|   |   |   |
-------------
|   |   |   |
-------------
```

For alle oppgaver, lag gjerne flere kall til funksjonen så man ser at den kan brukes flere ganger

LETTERE: Ignorer delen med input fra bruker, lag en funksjon som bare printer en helt identisk tabell hver gang (fast antall rader og kolonner med fast bredde)

tabell_v05.py

MIDDELS: Lag funksjonen slik figuren over indikerer. Dvs. basert på tall brukeren har gitt inn, skal kallet i skriptet gi argument til funksjonen, og funksjonen må ha parametre som gir mulighet til å regulere størrelsen på tabellen.

tabell_v1.py

VANSKELIG: gjør MIDDELS + (a) endre funksjonen så en av parametrene gis en default-verdi, og vis et kall hvor dette benyttes til å droppe argument for den parameteren.

(b) Endre program og funksjon så kolonnebredde blir en global variabel i stedet for en parameter.

tabell_v2.py

Oppsummering

- Funksjoner gir mange fordeler:
  - Bedre struktur, arbeidsdeling, kortere programmer, gjenbruk
  - Særlig hvis like eller lignende oppgaver skal gjøres mange steder
- En funksjon må defineres og består av hode og kropp:

```
def funksjonsnavn():
    kode...
```

- En funksjon kalles (kjøres) med funksjonsnavnet:

```
funksjonsnavn()
```

- Lokale variable lurt for interne beregninger i funksjonen
  - Plass frigjøres igjen straks funksjon er slutt
- Globale variable mindre lurt (unntatt konstanter)

Oppsummering

- Funksjoner kan ta imot verdier i parametre.
  - en variabel som tar imot en verdi når funksjonen blir kalt:

```
def funksjon(param1, param2):          # Variable
    kode...
```

- Verdier kan overføres til funksjoner ved hjelp av argumenter:

```
funksjon(argument1, argument2)         # Verdier
```

  - Vanligst: posisjonelle argumenter
    - rekkefølge av argumenter matcher parameterlista i funksjonshodet
  - Men kan avvike
    - keyword-argumenter: muliggjør annen rekkefølge
    - default-verdier: gjør at man kan droppe parametre