Programmering

Kursmål Del 3

- Sannhetsverdi og tester
- Vilkår (if, elif, else)
- Repeterende kode, løkker
- While løkker
- For løkker
- Funksjoner
- Lister
- Hente ut elementer fra en liste
- Feilsøking og hjelp
- Kodestil
- Oppgaver grunnleggende programmering

3.1 Sannhetsverdier og tester

3.1.1 Sannhetsverdier

Vi bygger opp utsagn med disse binære relasjonene:

Relasjon	Betydning
==	Er lik
!=	Er ikke lik
<	Er mindre enn
<=	Er mindre enn eller lik
>	Er større enn
>=	Er større enn eller lik

Python har to sannhetsverdier, True og False.
Alle utsagn evalueres til sann eller usann og får derfor henholdsvis verdien True eller False.

Vi prøver ut relasjonene i konsollen:

```
In [1]:  3 < 2
Out [1]: False

In [2]:  3 > 2
Out [2]: True

In [3]:  3 != 2
Out [3]: True

In [4]:  "matematikk" == "matematikk"
Out [4]: True
```

Vi kan bygge opp større utsagn med konjugasjonene «not», «and» og «or»:
Vi prøver ut konjugasjonene i konsollen:

```
In [1]:  3 < 2 and 3 > 2
Out [1]: False

In [2];  3 < 2 or 3 > 2
Out [2]: True

In [3]   not(3 < 2)
Out [3]: True
```

3.1.2 Vilkår (if-tester)

Vi lager en test i Python med kontrollordet «if», etterfulgt av et utsagn og et kolon. Så kommer kode som skal utføres hvis utsagnet evalueres til True. Dette kan kombineres med kontrollordet «else», et kolon og til slutt en kode som skal utføres hvis utsagnet evalueres til False.

Kontrollordene if, elif og else skal ikke være innrykket, men resten av if- koden skal være innrykket ett nivå.

Eksempel:

```
n = 10
a = 30

if  a > n:
        print(f "{a} er større enn {n} ")
else:

        print(f "{a} er ikke større enn {n} ")
```

I eksempelet over blir output 30 er større enn 10, siden verdien av variabelen «a» er større enn verdien av variabelen «n». Da blir utsagnet « a > n » tolket til True, og linje 4 blir utført.

Spyder og andre editorer sørger i utgangspunktet for riktig innrykk automatisk, men det kan være nødvendig å rette opp manuelt. Det lønner seg å lage innrykk med tabulatortasten, til venstre for Q på tastaturet. Da blir det minst feil. Innrykkene må være like på hvert nivå, så forsøk å unngå å bruke mellomromstasten.

Når du skal gjennomføre flere tester etter hverandre, kan du også bruke kontrollordet «elif», en forkortelse for «else if».

Eksempel:

```
tall = int(input("Skriv inn et tall: "))
if tall % 3 == 0:
        print(f"{tall}er delelig med 3. ")
Elif tall % 3 == 1:
        print(f"{tall}er ikke delelig med 3. Divisjonen har 1 i rest. ")
else:
        print(f "{tall} er ikke delelig med 3. Divisjonen har 2 i rest.")

Skriv inn et tall: 35
35 er ikke delelig med 3. Divisjonen har 2 i rest.
```

Det er ikke nødvendig å ha med «else». Det går fint å bruke en if-test uten «elif» eller «else».

Eksempel:

```
a = int(input("Skriv inn et heltall:  "))
svar = "Variabelen a er større eller lik 5. "

if  a < 5:
        svar = "Variabelen a er mindre enn5. "
print(svar)
```

```
Skriv inn et heltall:  34
Variabelen a er større enn eller lik 5.
```

```
tall_1 = int(input('Skriv inn et heltall: '))
tall2 = int(input('Bra, skriv inn et nytt heltall: '))
if tall1 > tall2:
   print(f'{tall1} er større enn {tall2}')
else:
   print(f'{tall1} er ikke større enn {tall2}')
```

Oppgaver

17. Skriv et program som ber brukeren om et tall og der etter skriver en melding om tallet er positivt eller negativt.

18. Lag et program som ber brukeren om to tall og sjekker om tallene er like. Programmet skal skrive resultatet til skjerm.

19. Skriv et program som ber brukeren om et heltall og deretter svarer om det er et oddetall eller partall.

20. Lag et program som ber brukeren om to tall og sjekker om tallene har ulike fortegn. Programmet skal skrive resultatet til skjerm.

3.2 Repeterende kode, løkker

Datamaskinens fortrinn kommer tydelig til syne når vi ser hvordan vi kan gjennomføre kodegjentakelse. Vi skal se på to måter å programmere repeterende koder:

for-løkker og while-løkker.

3.2.1 While-løkke

Syntaksen for en while-løkke er kontrollordet «while», etterfulgt av en logisk test, altså et utsagn som blir evaluert til True eller False, og et kolon. Instruksjonene i while-løkka blir utført så lenge utsagnet blir evaluert til True. Alle instruksjoner i løkka skal rykkes inn ett nivå.

Eksempel

```
n = 0
while n < 4:
        print(n)
        n = n+1
```

Skriver ut:
0
1
2
3

For hver gang løkka utføres, skrives verdien av n ut, så øker verdien av n med 1 helt til n ikke lengre er mindre en 4.

I eksempelet over starter variabelen n på 0. For hver gang løkka utføres, skrives verdien av n ut, og så økes verdien av n med 1. Dette skjer helt til verdien av n ikke lenger er mindre enn 4.

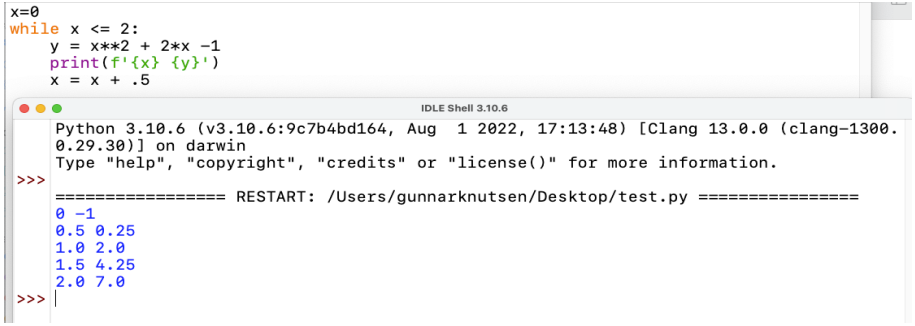
Dette kan vi eksempelvis bruke til å lage verditabell for en funksjon:

```
x = 0
while x <= 2:
    y = x**2 + 2*x -1
    print(f'{x} {y}')
    x = x + 0.5
```

Koden ovenfor starter x på 0. Så lenge x er mindre enn eller lik 2 utføres følgende:

– y settes til $x^2 + 2x - 1$.
– x og y skrives til skjerm.
– Variabelen x økes med 0,5.

Når vi kjører koden, får vi:



I programkoden for verditabellen ovenfor er de forskjellige tallverdiene angitt direkte i koden der de trengs. Koden blir ryddigere og derved lettere å lese og vedlikeholde om vi strukturerer den litt annerledes: Vi samler variabler og konstanter øverst i koden, og refererer til dem senere, vi legger dem globalt, se eksempelet nedenfor.

Eksempel
Lag en verditabell for f (x) = x2 + 2x = 1 for x ∈[0,2] med steglengde 0,5.
Vi bruker while- løkke til å styre x-verdiene. Vi samler variablene i starten av koden, globalt. Vi runder av verdiene når vi skriver dem ut.

```
xstart = 0      #Nedre grense for x
xslutt = 2      #Øvre grense for x
xsteg = 0.5     #Steglengde i verditabell

x = xstart

while x <= xslutt:
        y = x**2 + 2*x -1
        print(f"{x:.2f}  {y:.2f}")
        x = x + xsteg
```

While-løkker må ofte kombineres med en eller flere tellere eller midlertidige variabler. Vi tar med et eksempel hvor svaret på oppgaven er antall ganger løkka har kjørt.
I tillegg vil det være flere eksempler og oppgaver med while- løkker i oppgavene til denne delen. En feil i en while-løkke kan lett føre til en evig løkke. Da må du manuelt avbryte programmet ved å trykke på **stopp-knappen i øvre del** av konsollen, og så rette feilen.

Eksempel
Lag et program som løser likningen $5^x = 625$ ved å telle hvor mange ganger 625 er delelig med 5.

Grunnlaget starter med n på 625 og deler med 5 helt til det ikke går lenger. Og så teller vi underveis hvor lenge det går.

Vi gir variabelen n startverdien 625, se linje 1. Deretter oppretter vi en teller- variabel, se linje 2. Vi kjører while- løkka så lenge n er delelig med 5, altså har rest null om vi hadde delt på5. Inni løkka deler vi n på 5 og lar ny n ha denne verdien. Så øker vi teller med en.

```
n = 625
teller = 0
While n % 5 == 0:
        teller  += 1
        n //= 5
print(teller)
```

Programmet gir 4 til svar, som betyr at $5^4 = 625$

Oppgaver

21. Skriv et program som lager verditabell for $f(x) = -x^2 + 7x - 12$ med steglengde 0,1 for x [-10, 10].
22. Skriv et program som skriver ut fem- gangen.
23. Skriv et program som skriver ut leddene i en tallfølge $\{a_i\}$, definert ved:

    $a_i = 4$

    $a_n = 2 * a_{n-1} + 1$

    Programmet skal slutte når a_i overstiger 1000.
    Programmet skal altså skrive ut tallene 4, 9, 19, 39, 79, 159, 319 og 639

24. Skriv et program som bruker en teller til å finne svaret i regnestykket 15 -7, uten å bruke operatoren minus.
    (Tips: Programmet skal bruke en løkke til å telle seg opp fra 7 til 15. Svaret skal vise at løkka har kjørt 8 ganger.
25. Skriv et program som løser likningen $2^n = 128$ ved å telle hvor mange ganger vi må opphøye 2 for å få 128.
26. Lag et program som skriver ut en hilsen, spør om brukeren vil se hilsenen en gang til, og fortsett til brukeren svarer nei.

3.2.2 For-løkke

Når vi på forhånd ikke vet hvor mange ganger en løkke skal kjøre, må vi bruke while-løkke.
Når vi vet hvor mange ganger koden skal gjentas, bruker vi en for-løkke.
Syntaksen er kontrollordet «for» etterfulgt av en fritt valgt variabel, kontrollordet «in» og til slutt et objekt som kan itereres over, for eksempel en mengde, og et kolon.
Alle instruksjoner i løkka skal rykkes inn ett nivå. For løkke er normalt kalt «teller løkke»

Vi lager indeksmengder med kommandoen range. Kommandoen range(3, 8) betyr heltallene fra og med 3 til, men ikke med, 8.

Eksempel

```
For i in range(3, 8):
        print(i)
```

Utskrift:

```
3
4
5
6
7
```

Hvis du utelater første argument i range, antar Python at første argument er 0. Dersom du legger til et tredje argument, angir dette steglengden i intervallet.

Kommando	Verdi
range(0, 5)	0, 1, 2, 3, 4
range(5)	0, 1, 2, 3, 4
range(2, 5)	2, 3, 4
range(0, 5, 2)	0, 2, 4
range(10, 5, -1)	10, 9, 8, 7, 6

Indeksen trenger ikke å brukes inni løkka. Da brukes range til å bestemme hvor mange ganger løkka skal utføres.

```
for i in range(3):
print("Matematikk")
```

Resultat:

Matematikk
Matematikk
Matematikk

Vi har tatt med et eksempel på en løkke som lager verditabell.

Eksempel

Vi lager en verditabell for f(x) = 3 x 2 + 1

```
print ("x  f(x)")
print ("-------")
for x in range(0, 5, 2):  # start, tall opp til, stegverdi
    print(f"{x}  {3*x**2 + 1}")
```

Resultat

x	f(x)
0	1
2	13
4	49

For-løkker kan også brukes med andre objekter enn de vi lager med range.
Noen datatyper, som strenger og lister, kan man iterere over direkte.

Eksempel

```
For bokstav in "søt":
    Print(bokstav)

    Resultat
    S
    ø
    t
```

Oppgaver

27. Lag et program som skriver ut seks- gangen
28. Lag et program som skriver ut alle oddetallene mellom 0 og 100.
29. Skriv et program som skriver ut de 100 første leddene i den geometriske følgen{3, 6, 12, 24….}.
30. Skriv et program som gir denne outputen:

```
x
xxx
xxxxx
xxxxxxx
```

Tips: 'x'*4 gir xxxx.

3.3 Funksjoner

Funksjoner i programmering er en måte å samle kode i en sammensatt bolk, slik at det blir lettere å lese, skrive og vedlikeholde koden. Du kan klare deg uten funksjoner, men det er lettere å programmere med funksjoner enn uten.

Syntaksen for funksjoner er kontrollordet def, deretter navnet på funksjonen, to parenteser og et kolon. Dersom du vil at funksjonen skal kunne ta et eller flere argumenter, så skriver du dem inn i parentesene.

Koden nedenfor definerer $f(x) = x^2 + 5x - 12$:

```
def f(x):
    return x**2 + 5*x – 12
```

Vi bruker funksjonen på vanlig måte, for eksempel f(2), altså ved å skrive navnet, med eventuelle argumenter. Hvis vi skal ta vare på verdien fra funksjonen, legger vi verdien inn i en variabel. Vi kan klare oss uten funksjoner, men funksjoner gir ryddigere og enklere kode, som for eksempel i verditabellen på neste side.

Eksempel

Lag en verditabell til $f(x) = x^3 - x^2$ for $x \in [0,1]$ med steglengde på 0,2.

```
def f(x):
        return x**3 – x**2
xstart = 0
xslutt = 1
xsteg = 0.2

x = xstart
y = f(x)
while x <= xslutt:
        print(f"{x:.3f} {y:.3f}")
        x = x + xsteg

        y = f(x)
```

0.000	0.000
0.200 -	0.032
0.400 -	0.096
0.600 -	0.144
0.800 -	0.128
1.000	0.000

Funksjoner kan ha flere argumenter. Da skiller vi argumentene med komma i definisjonen.

Funksjoner kan også defineres uten noe return-utsagn. Når du bruker en slik funksjon, blir alle funksjonens linjer med kode utført, men funksjonen returnerer ingen verdi.

Eksempel

Vi skal lage en funksjon som legger sammen to tall.

```
def addisjon (n, m):
        return n + m

summen = addisjon(5, 7)
print(summen)
```

resultat 12

31. Lag en funksjon f som returnerer det samme som den matematiske funksjonen $f(x)=3x^2+3$. Bruk deretter f til å skrive ut f(0), f(-3) og f(5).

32. Skriv et program som lager verditabell for $f(x) = x^2 - 7x + 1$ med steglengden 0,1 for $x \in [-5, 5]$. Bruk f som funksjon underveis i programmet.

33. Lag en funksjon som returnerer produktet av to tall.

34. Lag en funksjon som adderer to tall og skriver ut regnestykket og svaret.

3.4 Lister

3.4.1 Å lage lister selv

Lister er elementer adskilt med komma mellom hakeparenteser, for eksempel [1, 2, 3] og ['Maria', 'Siri', 7].

Du oppretter en tom liste ved å tilordne verdien [ ] til en variabel, for eksempel liste = [ ]. Når vi skal legge til elementer i en liste, bruker vi kommandoen append. Denne bruker vi med listevariabelen som forstavelse, adskilt med et punktum.

Eksempel

```
liste = []
liste.append(5)
liste.append(2)
liste.append('a')
print(liste)
```

Resultat

```
[5, 2, 'a']
```

Når vi skal bygge opp en tallfølge i en liste, bruker vi løkker til å legge tallene inn i lista, se x og y verdi(append) i eksempelet nedenfor.

Eksempel
La f være funksjon gitt ved $f(x) = -0{,}4x^2 - 2$
Skriv et program som legger x-verdien og y-verdien fra verditabellen til f inn i to lister. Velg x mellom -2 og 2 med steglengde 0,5.

```
def f(x):
        return -0.4*x**2 – 2
xstart = -2
xslutt = 2
xsteg = 0.5

xverdier = []
yverdier = []

x = xstart
y = f(x)

while x <= xslutt:
        xverdier.append(x)
        yverdier.append(y)
        x = x + xsteg
        y = f(x)
print(xverdier)
print(yverdier)
```

Alle kommandoer vi bruker på lister angis med listevariabelen som forstavelse.

Resultat

[-2, -1.5, -1.0, -0.5, 0.0, 0.5, 1.0, 1.5, 2.0]
[-3.6, -2.9, -2.4, -2.1, -2.0, -2.1, -2.4, -2.9, -3.6]

3.4.2 Hente ut elementer fra en liste

Lister er itererbare objekter, så vi kan løpe gjennom dem med en for-løkke.

Eksempel

```
liste = [1, 2, 3, 4]
for tall in liste:

        print(tall)
```

Utskrift

```
2
3
4
```

Vi plukker ut bestemte elementer fra en liste med hakeparenteser. liste[0] gir første element, liste[1] gir andre element, liste[2] gir tredje element og så videre. liste[-1] gir siste element. For å plukke ut en lengre del av listen, bruker vi intervallnotasjon med kolon, for eksempel liste[2:4]. Vi teller fra 0, så dette betyr tredje og fjerde, men ikke femte element.

Her er noen eksempler på hvordan vi kan plukker ut elementer fra en liste. Vi tar utgangspunkt i lista definert som liste = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h']:

Kommando	Resultat	Forklaring
liste[0]	'a'	Første element er nr. 0
liste[0:3]	['a', 'b', 'c']	Tar ikke med nr. 3 (fjerde)
liste[:3]	['a', 'b', 'c']	Tar med element på plass nr. 0, 1 og 2
liste[-1]	'h'	Siste element er nr. −1
liste[3:6]	['d', 'e', 'f']	Fra og med nr. 3 til nr. 6
liste[3:-1]	['d', 'e', 'f', 'g']	Fra nr. 3, tar ikke med siste element
liste[3:]	['d', 'e', 'f', 'g', 'h']	Fra nr. 3, tar ikke med siste element

Vi tar også med kontrollordet len, som gir oss lengden av listen, altså antall elementer i listen.

```
liste=[6,2,5,3,5,6,6] #  en liste med sju elementer
print(len(liste))
```

Resultatet her blir: 7, fordi lista inneholder sju elementer.

Eksempel

```
liste = ['a', 'b', 'c', 'd']
for bokstav in liste:
print(bokstav)
```

Resultatet blir

```
a
b
c
d
```

Vi kan bruke len-kommandoen sammen med en løkke til å gjentatte ganger velge ut bestemte elementer i lista. Kommandoen len(range()) lager en indeksmengde med like mange elementer som lista har.

Eksempel

Regn ut differansen mellom de seks første kvadrattallene.

Len(liste) gir antall elementer i lista, mens range(len(liste)) gir en indeksmengde med samme lengde. Antall differanser er en mindre enn antall elementer i lista, så for- løkka har indeksmengde range(len(liste) – 1).

```
liste = [1, 4, 9, 16, 25, 36]
for i in range(len(liste) – 1):
        differanse = liste[i + 1] - liste [i]
        print(differanse)
```

Resultat

```
3
5
7
9
11
```

35. Sett opp en liste som består av tallene 0–9. Skriv lista til skjerm.
36. Opprett en tom liste. Velg tre navn som du legger i lista. Skriv lista til skjerm.
37. Opprett en tom liste. Legg til noen elementer i lista. Skriv ut hvor mange elementer det er i lista.
38. Sett opp en liste med ti ulike elementer. Plukk ut det tredje, fjerde og femte elementet. Lagre elementene i hver sin variabel og skriv dem til skjerm.
39. En liste er gitt som: [1, 4, 9, 16, 25, 36]. Bruk en for-løkke til å regne ut produktet av alle elementene i lista.
40. Skriv et program som lager en tom liste, ber brukeren om et heltall og så lar en while-løkke legge til 1-tall i lista, helt til lengden av lista er lik tallet brukeren la inn.
41. Skriv et program som definerer funksjonen f(x)=x2+3x−1. Programmet skal be brukeren om en og en x-verdi og legge dem inn i en liste. Når brukeren avslutter, skal programmet regne ut de tilhørende funksjonsverdiene og legge dem inn i en egen liste. Til slutt skal de to listene skrives til skjerm.

3.5 Feilsøking og hjelp

Alle som skriver programmer, opplever feil. Det er derfor nyttig å venne seg til å lese feilmeldinger. I feilmeldingen får du beskjed om hvilken linje feilen er på, en beskrivelse av feilen og hvilken type det er. Med litt trening, får du god hjelp i programmeringen av feilmeldingene.

De vanligste feilene er disse:

- Syntax Error: Du har ikke overholdt rettskrivningsreglene. Har du glemt kolon? Har du noen ekstra tegn? Har du brukt parenteser feil? Er det feil ved innrykk? Ofte er feilen på linja over den feilmeldingen sier at feilen er på.
- IndexError: Vanligvis betyr dette at du prøver å plukke ut et element fra en liste, og så har du gjort en tellefeil i starten eller slutten av lista. Sjekk nøye hvor mange elementer det er i lista og hvilke elementer du skal plukke ut.
- NameError:Feilsomoftegjeldervariabler.Kanskjeduharstavetvariabe- len feil? Eller glemt å opprette variabelen før du bruker den?
- TypeError: Du bruker feil datatype i en funksjon, for eksempel bruker et desimaltall som argument i range.

Når du feilsøker, kan det være god hjelp å se hvilken verdi variablene dine har underveis når programmet kjører. Dette får du til ved å legge inn print- kommandoer på utvalgte steder. I tillegg kan du se på «Variable Explorer» i Spyder, som viser verdien av variablene etter at programmet er ferdig.

Når du er litt usikker på hva koden din egentlig gjør, kan du teste deler av den i konsollen, slik vi gjør i eksemplene 12, 13 og 29.

Hvis du ikke finner feilen, kommer du ofte langt med å google «Python» + feil-meldingen.

Du finner mye god dokumentasjon av Python på nettet. Et fint sted å starte å se etter hjelp er The Python Tutorial, lagt inn som lenke på siden. Hvis det gjelder plotting, kan du lese om dette på Matplotlib sine sider, lenken er lagt til. . Hvis det gjelder det NumPy, kan se dette på numpy.org sine hjemmesider, også lagt til i lenkene.

3.6 Kodestil

Etter hvert som vi begynner å beherske de grunnleggende elementene i programmering, kan det være på sin plass å bruke litt tid på kodens utseende. Da kan du lese PEP8 (på deres hjemmeside, lenken er lagt til. Dette er den offisielle anbefalingen for hvordan Python kode bør se ut.

Følger du anbefalingene i PEP8, så øker lesbarheten av koden. Som lærere kommer vi til å lese mye kode, så det er en fordel for oss om både kolleger og elever skriver lettlest kode.

3.7 Oppgaver i grunnleggende programmering

Du har nå vært gjennom deler av grunnleggende programmering. Nå kan du øve på mange ulike problemstillinger i programmering. Her er noen oppgaver:

Oppgaver

42. Skriv et program som legger partallene mellom 0 og 100 inn i en liste
43.

a Lag en funksjon gjennomsnitt(a, b) som returnerer gjennomsnittet av a og b.
b Lag et program som for hver av partallene mellom 1 og 10 skriver ut gjennomsnittet av tallet og det påfølgende oddetallet. I konsollet får du

2.5
4.5
6.5
8.5

44. La f være funksjonen gitt ved: $f(x) = x^2 + 2x + 3$
    a. Lag en verditabell for f(x). Velg selv grenser og steglengde.
    b. Utvid programmet ditt til å la bruker velge minste og største verdi for x og trinnene i tabellen.
45. Skriv et program som ber brukeren om et heltall n og så summerer tallene {1, 2, 3…n}.

46. Skriv et program som ber brukeren om et oddetall n, sjekker at n er et oddetall og summerer tallene {1, 3, 5…n}.
47. Lag et program som ber om et heltall n og skriver ut alle tall i 6- gangen som er mindre en n.
48. Lag en funksjon som bruker operatoren * og en løkke til å regne ut x^n, uten å bruke **.
49. Skriv et program som ber brukeren om et tall og så skriver til skjerm hvor mange siffer tallet har.
50. Lag et program som skriver ut de 100 første Fibonaccitalene
51. Skriv et program som gir denne outputen:

```
x
xxx
xxxxx
xxxxxxx
```

52. Skriv et program som lager liste over primtall mellom 0 og 100.
53. Lag et program som ber brukeren om et tall og så finner tallets største primtallsfaktor.
54. Be om et heltall n og finn neste perfekte tall, altså minste tall større en n som er slik at tallet er summen av alle divisorer som er mindre enn n.

KLAR FOR NESTE MODUL?