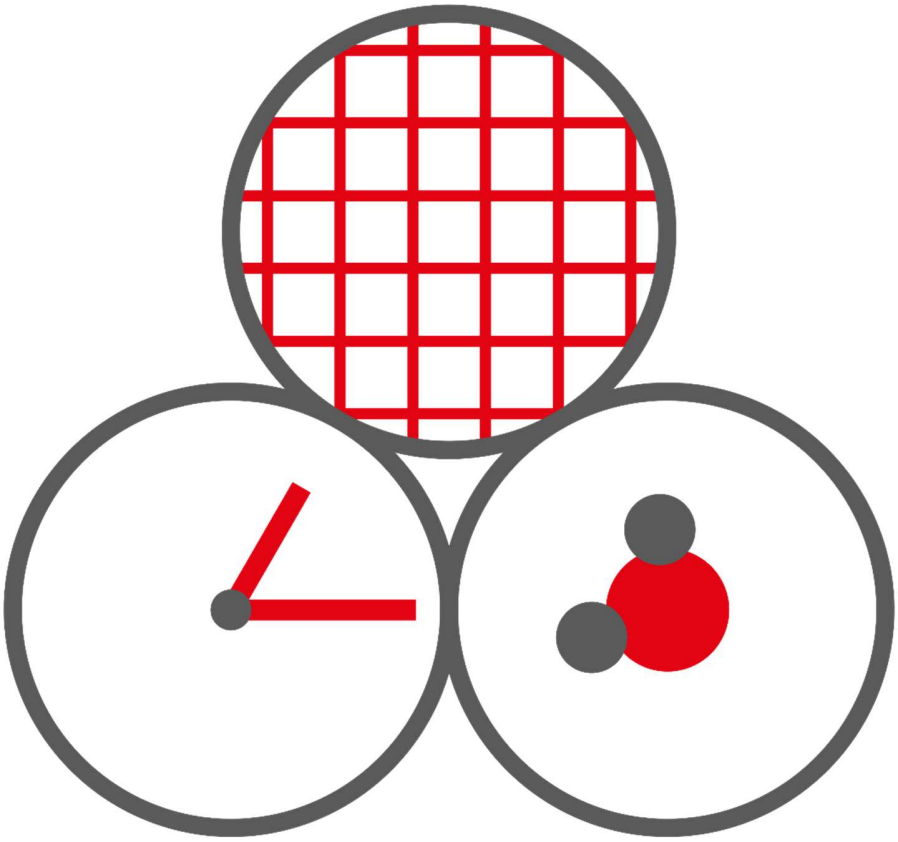
Intro til Programmering

Naturfag

Matematikk 1P



Erik Skaar

Hellerud VGS

Om dette arbeidsheftet

Dette heftet er ment som et arbeidshefte til elever på Hellerud videregående. Arbeidsheftet sikter på å gi en grunnleggende introduksjon til programmering både i form av tenkemåter og programmeringskunnskap.

Når du skal programmere, så trenger du et verktøy hvor du kan **lage** et **program** Dette gjør du ved å skrive **koder**, og deretter kjøre **programmet**. Et **program** er altså en sum av **koder**. Det finnes tre gode alternativer for dette:

1. **Nettbaserte løsning:**

   Det er mange ulike nettbaserte løsninger, men et godt alternativ er https://trinket.io/python3. Dette er løsningen som dette arbeidsheftet kommer til å basere seg på. Fordelene med en nettbasert løsning er at man ikke trenger å installere noe, men ulempen er at det er vanskeligere å *lagre* programmer som man utvikler.

2. **IDE – Integrated Development Environment:**

   Om du har et større behov for programmering i faget, så anbefales det å installere Spyder (https://www.spyder-ide.org/). Fordelen med en IDE er at den kommer med alt du trenger. Ulempen er at det kan være overveldende og man trenger tid til å venne seg til programvaren.

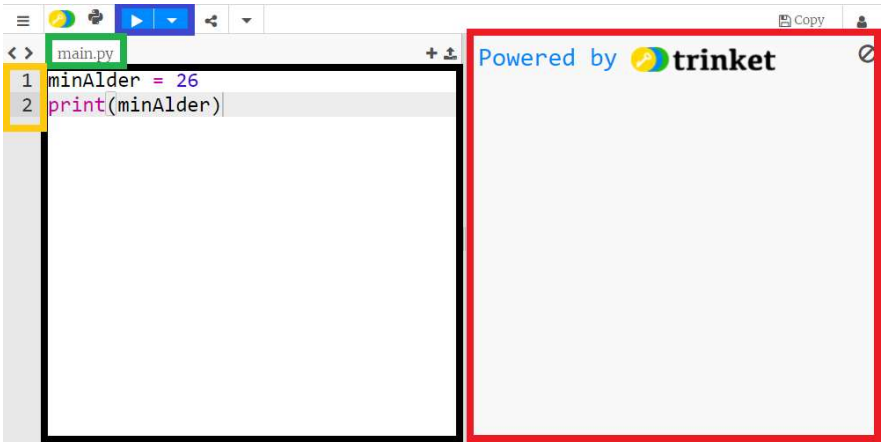
3. **Tekst-editor og terminalen:**

   Dersom du har et stort ønske om å programmere på samme måte som profesjonelle program- og apputviklere, kan du vurdere en tekst-editor og en terminal. Det anbefales at du starter med en tekst-editor som Sublime Text og beveger seg over til Visual Studio Code etter hvert. Du må også installere Python. Fordelen med denne løsningen er at du kan gjør alt som går an og at du fremstår som en profesjonell programmerer. Verdt bryet? Nei, ikke for fagene 1P og naturfag..

Introduksjon til trinket.io/python3

Trinket.io/python3 er et nettbasert tekstprogram for programmeringsspråket Python, versjon 3. Ved å bruke en nettbasert løsning vil resultatet i de fleste tilfeller bli helt likt som om du kjørte programmet i en tekst-editor. Derfor skal du bruke *Trinket.io/python3* i dette kurset.

Først skal du bli kjent med *Trinket.io/python3* som verktøy:



Hurtigtaster for å lagre koden:

Trykk i den svarte boksen og trykk deretter CTRL+A

Trykk deretter CTRL+C

Åpne et Word-dokument og trykk CTRL+V

Det **svarte** området er der du skriver koden din. Det du skriver i dette området er det som blir kjørt av trinket.io, når du kjører programmet ditt.

Det **blå** området er knappen hvor du kan be trinket.io om å kjøre programmet ditt. Trykk på PLAY-knappen, og programmet ditt blir kjørt

Det **røde** området er der resultatet ditt blir vist. I eksempelet over ville tallet 26 bli vist om jeg kjørte programmet. Vi skal se mer på dette senere.

Det **gule** området er området som viser linjenummeret til hver kodelinje. Dette er veldig nyttig for å finne feil. Python sier nemlig hvilken linje som er feil om du har feil i koden.

Det **grønne** området er navnet til filen din. Dette trenger du aldri å forandre, men vit at trinket.io kjører filen som heter main.py. Det vil si at det er viktig at det står main.py her.

Merk: dersom du vil lagre koden din, så kan du kopiere koden fra den svarte boksen og lagre den i et Word-dokument.

Innhold

Forfatteren skriver: hvorfor skal du programmere?

Hvorfor skal du og dine medelever lære å programmere? Jeg skal prøve å gi et generelt svar på dette, uten å svare «fordi UDIR har bestemt at du skal lære det på skolen».

Det året du fylte seks år startet du på skolen. Du lærte om naturen, religioner, historie, kroppen din osv. Dette er kunnskap som er nyttig å ha for å kunne fungere i samspill med resten av samfunnet. Du lærte også ferdigheter som å skrive, lese og regne. Dette er ferdigheter som muliggjør samarbeidet mellom mange mennesker. Hvis vi ikke hadde kunnet lese og skrive, så måtte vi husket alt som vi noensinne ble fortalt og det hadde blitt et kaos når vi skulle samarbeide om større oppgaver. Dette hadde også vanskeliggjort informasjonsoverføring mellom mennesker.

Om noen hadde sagt at vi skulle møtes på den tredje dagen etter det hadde regnet når solen står på sitt høyeste punkt, så kunne man tenke seg at tre personer kunne tolket dette ganske forskjellig.

Det at vi kan skrive, lese og regne gjør at komplekst samarbeid er mulig. I dag er verden blitt mer komplisert og mye av hverdagen og samarbeidet er dominert av informasjon. Informasjon kalles også data, og med programmering får du et verktøy som kan behandle all denne informasjonen på en ekstremt effektiv måte. Det kan hende at du spør deg selv, hvorfor bruker jeg ikke bare Excel? På samme måte som gangetabellen er en effektivisering av enkel multiplikasjon og at matematikk er effektivisering av logisk tekning, så er Excel effektivisering av tabeller[1], mens programmering er effektivisering av all data og også bruk av dataen. Personer som kan programmere kan dermed gjøre ting de som bruker Excel blir svette av å bare tenke på.

[1] Teknisk sett er det et programmeringsspråk i Excel, så du kan bruke Excel til ekstremt mye.

Videre kan det hende at du tenker: jeg skal ikke holde på med realfagene, hvorfor skal jeg lære meg dette? Det kan hende at du ikke trenger programmering til noe i løpet av livet. Akkurat som det kan hende at du at du tror at du ikke trenger matematikk til noe igjennom livet. Men, det er ganske stor sannsynlighet at programmering kunne gjort flere ting i livet ditt enklere. Jeg har for eksempel laget et budsjettprogram som enkelt lar meg holde oversikt over økonomien min (som kanskje ikke høres så kult ut, men vent til dere betaler regninger og lån!). Programmering kan også brukes til mange ting som ikke har noe som helst med realfagene å gjøre. Logoen som du så på første side er skrevet i Python. Videre er alle sidene på nettsidene mine skrevet med en underliggende programmeringslogikk som vi skal lære mer om senere i dette heftet[2].

Når du skriver koder tvinges du til å tenke grundig gjennom hva du ønsker at programmet skal utføre. Dette gjør at du får en dypere faglig forståelse enn dersom du lar ExCel eller en kalkulator gjøre de samme beregningene.

Kanskje har du veldig lyst til å lage et spill eller designe noe digitalt, eller «hacke» en nettside. For at du skal kunne gjøre noe av dette, så må du ha gode grunnleggende ferdigheter innenfor programmering. Det meste av det du lærer i dette heftet om Python gjør det lettere å lære andre programmeringsferdigheter.

Det er avgjørende å ha en ha en forståelse for de grunnleggende prinsippene i programmering. Dette heftet prøver å gi deg denne kunnskapen. Heftet har med andre ord to fokus-områder. Det første er å lære deg som bruker heftet det mest grunnleggende innenfor programmering, slik at du forstår hva programmering er og kan lære deg mer om du vil. Det andre fokus området er å gjøre deg klar for programmering i videregående, slik at du kan løse alle programmeringsoppgavene som du møter i løpet av skolen og senere studier.

Heldigvis går disse to målene som hånd i hanske, altså har du lært det ene så har du lært det andre.

[2] https://eskaar.no/

Intro til programmering

Hva er kode?

Kode er å gi instrukser til PC'en, som den kan forstå og utføre. Programmeringsspråket vi skal bruke heter Python og er bygget for å være et lettere (og morsommere) programmeringsspråk. En av de første tingene vi skal lære i Python, er hvordan du får en PC til å huske **verdier**. Med **verdi** mener vi tall, bokstaver eller en liste.

Lagre informasjon i Python – Variabler

For at Python skal kunne huske en verdi må vi lagre den i det som kalles en **variabel**. Det betyr at vi gir verdien vår et navn. Eksempel: vi ønsker å lagre **verdien** 2021 i en variabel som heter årstall. For å lagre **verdien** bruker vi «er lik»-tegnet (=).

I matematikk bruker vi = for å beskrive at høyre siden og venstre siden er like, men i programmering bruker vi = for å *definere* høyre siden til å være venstre siden. Om vi skriver følgende kode i Python, så lagrer vi tallet 2021 i en **variabel** som heter årstall:

```
årstall = 2021
```

Det kan hende at vi ønsker å lagre navnet vårt i en **variabel**. Vi må bruke "-tegnet for å kunne skrive tekst i Python. For å lagre navnet ditt i en **variabel** med navnet «navn», så kan du skrive koden under:

```
navn = "Erik"
```

For å lagre desimaltall i Python, så må vi bruke «.» (punktum) istedenfor «,» (komma) som vi bruker på norsk. Under kan du se et eksempel på hvordan vi skriver tekst, tall og desimal tall som verdier til variabler i Python.

```
tekst          tall          desimaltall
"tekst"          123               123.456
```

Du kan kalle variabelen din for nesten hva som helst, men det er noen regler for navnene du kan bruke:

1. Ingen spesialtegn, som ,.-§!+ eller lignende
2. Ingen tall i starten av variabelnavnet
3. Ingen ord som blir brukt av Python. Disse ordene blir farget når du skriver dem i trinket.
    - Eksempler: print, if, else, for, while, min, max

Variabelnavnene i venstre tabell er godkjente variabelnavn, mens navnene i høyre tabell er ikke godkjente variabelnavn i Python:

```
GODKJENT

variabel = 1
vårData = 2
mittNavn = "Erik"
tekst2 = "tekst"
teller = 4
```

```
IKKE GODKJENT

1teller = 5
print = 6
+mittNavn = "Erik"
!tekst = "tekst"
verdi.tall = 10
```

Oppgave 1

Hvilke av variabelnavnene er **IKKE** godkjente variabelnavn i Python?

```
1teller = 123      teller = "riktig"    tekst.2 = "tekst"
!teller = 1        Teller = 1           x = 10
xyz = 2021         alder = 17           navn = 100
```

Oppgave 2

Hva er reglene for variabelnavn i Python?

Oppgave 3

Hva mangler for at vi har lagret tallene 1, 2 og 3 i variablene tall1, tall2 og tall3

```
tall1 __ 1    ____ = 2    tall3 = __
```

Oppgave 4

I Python kan man bare skrive én variabel på hver linje[3]. Derfor vil den ene linjen som står i forrige oppgave ikke kjøre i Python. Skriv forrige oppgave på tre linjer slik at den kjører i Python.

Oppgave 5

Lag et program som lagrer navnet ditt i en variabel og alderen din i en annen variabel.

Oppgave 6

Hva kalte du variablene dine i forrige oppgave? Kunne du gitt variablene et mer informativt navn?

Utskrift i Python

Du har nå laget dine første programmer gjennom oppgavene 4, 5 og 6. Hvis du prøvde å kjøre disse på trinket.io, så skjedde det ingenting i **resultatområdet**. Det er fordi at dagens PC'er er de dummeste, mest lojale vennene du noensinne kommer til å ha. Den gjør nøyaktig hva du spør om, og ingenting annet. For at PC'en skal skrive ut resultatet ditt, så må du fortelle den det ved å bruke **funksjonen** print. Eksempelet under viser hvordan du kan skrive ut en tekstbit med print.

```
print("Erik")
```

Feilmeldinger

Før du kan begynne med oppgavene må du bli kjent med **feilmeldinger** i Python. En **feilmelding** er meldingen som dukker opp i **resultatfeltet** i trinket som forteller deg at noe er galt. Det er foreløpig kun én ting du skal lære deg om **feilmeldinger**.

[3] Dette er ikke helt sant. Se oppsummeringen for dette kapittelet.

I feilmeldingene dine står det først mange ord. Så kommer ordet «line» etterfulgt av et tall. Det tallet forteller deg hvilken linje i programmet ditt som er feil.

```
1 print("Erik"
```

```
  File
"/tmp/sessions/85a2e256bc848f80/main.py",
line 2
```

Av og til viser feilmeldingen ett tall høyere enn linjen som faktisk har feilen. I eksempelet over er det linje 1 som mangler en avsluttende parentes. Det er ikke linje to som mangler noe.

```
1 print("Erik")
```

```
Erik
```

Oppgave 7

Kjør programmet nedenfor på trinket.io/python3:

```
print("Hello World")
```

Oppgave 8

Hva er feil med koden under? *Hint:* Legg merke til at den ene parentesen er en annen farge.

```
print("Erik)
```

Oppgave 9

Hva mangler for at programmet skal lagre variabelen «navn» med verdien ditt navn? Forklar hva programmet gjør.

```
navn __ "Ditt navn"
print(navn)
```

Oppgave 10

Vi kan lage et program som skriver ut flere ting med å bruke «+» mellom variablene. Hva tror du programmet under skriver ut? Kjør programmet på trinket.io/python3.

```
print("Erik"+"26")
```

Oppgave 11

Fiks koden fra oppgave 10 slik at det blir et mellomrom mellom verdiene "Erik" og "26".

Oppgave 12

Skriv et program som lagrer alderen din og navnet ditt. Bruk så det programmet til å skrive ut en setning med begge verdiene.

Operatorer i Python – Pluss, minus, gange og dele

PC'er er veldig flinke til mange ting, men det de er best til er å regne. Vi skal nå lære å bruke Python som en kalkulator. For å kunne gjøre matematikk i Python skriver du regnestykket med symbolene +, -, * og / for henholdsvis pluss, minus, gange og dele.

	pluss	minus	gange	dele
Operator:	+	-	*	/
kode:	2+2	2-2	2*2	2/2
svar:	4	0	4	1

Legg merke til at om du lagrer en variabel som $5 \cdot 5$, så lagres variabelen som svaret til regnestykket, altså ikke $5 \cdot 5$, men 25 blir lagret i variabelen. Om det var uklart kan du prøve å kjøre koden under:

```
test = 5*5
print(test)
```

Det kan være verdt å merke seg at man kan bruke + og * på tekst. Om du skriver "erik"+" erik", så får man svaret "erik erik". Om man bruker * på tekst, så får man et litt annerledes resultat. Kjør koden under og se om du kan finne ut hva som skjer:

```
print("Erik "*5)
```

Oppdaget du at Erik blir skrevet ut 5 *ganger*? Dette skjedde fordi du brukte gange-operatoren med tallet 5 på teksten.

Siste nevneverdige bit med informasjon om matematiske operatorer i Python er at vi bruker ** for opphøyd i. Altså 5**4 betyr 5 opphøyd i 4, 5^4. Det vil være svaret til 5**4 som blir lagret, 625.

I oppgavene nedenfor og på neste side er det lurt om du først forsøker å tenke deg til svaret. Deretter kan du skrive koden inn i Trinket.io og kjøre programmet.

Oppgave 13

Hva gjør programmet nedenfor?

```
alder = 26
navn = "Erik"
print(alder*navn)
```

Oppgave 14

Hva mangler for at vi skal skrive ut hele navnet til Ola Nordmann?

```
print("Ola "__"Nordmann")
```

Oppgave 15

Hva er forskjellen på de to linjene under? Hint: Kjør kodene i trinket.io

```
print("Ola "+"Nordmann")
print("Ola ","Nordmann")
```

Oppgave 16

Skriv et program som skriver ut navnet ditt 5 ganger

Oppgave 17

Hva skrives ut?

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

Oppgave 18

Lag et program som lagrer tallene 7 og 9 i hver sin variabel og skriver ut 7+9, 7-9, 7*9 og 7/9.

Oppgave 19

Programmet under prøver å skrive ut 8*8 og 8*9. Hva er feil?

```
tall1 = 8
tall2 = 8
print(tall1*tall2)
print(tall1*tall2+1)
```

Oppgave 20

Forandre på koden fra forrige oppgave, slik at du ikke trenger å bruke +1 i utregningen. Hint: tall1 og tall2 er like.

Oppgave 21

Programmet under prøver å regne ut 10^{10}, men får 100 som svar. Hva tror du er problemet?

```
print(10*10)
```

Oppgave 22

Lag et program som lagrer x som 5 og printer ut 10^{x+1}. Svaret skal bli 1 000 000.

Ekstra – for spesielt interesserte

Input av data fra brukeren

Til nå har du selv laget verdier i et Python-program, men det er veldig nyttig å kunne spørre brukeren av programmet etter verdier. For at brukeren av programmet skal kunne skrive inn data så må vi bruke **input**. Se eksempel under:

```
innskrevetTekst = input("Skriv inn tekst her: ")
```

Koden over vil lage en variabel som heter «innskrevetTekst», og Python vil bruke input til å hente informasjonen. Det som står inni input sine parenteser, vil bli skrevet ut akkurat som om det skulle vært i en **print**-funksjon. Så pauser Python kjøringen av programmet for å vente på at du skriver det du vil skrive. Når du har skrevet det du vil, så trykker du Enter og Python fortsetter å kjøre programmet. Om vi legger til linjen print(innskrevetTekst), så vil det du skrev inn bli skrevet ut av Python etter du har trykket Enter.

Dessverre, så er det et lite problem med Python sin **input**-funksjon. Kjør koden nedenfor og se om du skjønner hva som er galt.

```
innskrevetTall1 = input("Et tall: ")
innskrevetTall2 = input("Et tall: ")
print(innskrevetTall1 + innskrevetTall2)
```

Python tolker all innskrevet data som tekst. Når du skriver inn tallene 5 og 7 i koden over, tenker ikke Python at dette er to tall, men at det er to tekster. Derfor setter Python tallene sammen til 57, istedenfor å legge sammen tallene slik at svaret blir 12.

Problemet løses ved å be Python behandle det innskrevne som tall ved å bruke funksjonene **int** (dersom input skal være heltall), eller **float** (dersom det skal tillates desimaltall). Dette kalles å **caste** variabler, og programmet ovenfor blir seende slik ut:

```
innskrevetTall1 = int(input("et tall:"))
innskrevetTall2 = input("et tall:")
innskrevetTall2 = int(innskrevetTall2)
print(innskrevetTall1 + innskrevetTall2)
```

Koden på forrige side viser to forskjellige måter å **caste** variablene på. Den ene er på samme linje og den andre er over to linjer. Det har minimalt å si hvilke av disse to skrivemåtene du velger. I programmering er stavemåten og tegnene vi bruker viktig. Det er ikke viktig om det er to linjer eller en.

Det siste du må få med deg om **casting** er at Python prøver uansett å **caste** inputet, så om du skriver tekst, så sier Python «jeg kan greie det». Spoiler, men Python greier det ikke og krasjer. Om du vil **caste** noe til tekst, må du bruke str().

Oppgave 23

Fullfør programmet under slik at en bruker kan lagre alderen sin:

```
alder = input(___
```

Oppgave 24

Hva gjør programmet?

```
tall1 = input("tall1:")
tall2 = input("tall2:")
print("sum",tall1+tall2)
```

Oppgave 25

Fiks programmet i forrige oppgave, slik at det skriver ut 11 når du skriver inn 5 og 6.

Oppgave 26

Skriv et program som leser inn tre tall. Programmet skal regner ut produktet av det første tallet og det andre tallet og regne summen av det andre og tredje tallet.

Kommentarer i Python

Om vi skriver store programmer så kan det fort bli veldig vanskelig å holde orden på hva som skjer i programmet. Vi kan beskrive de ulike kodene ved å bruke kommentarer. Kommentarer er en del av koden, men Python ignorerer den. Det vil si at hvis du skriver print(«Hei») inni en kommentar, så vil ikke Python skrive ut noe som helst, siden du spurte Python om å printe i en kommentar og Python ignorerer kommentarer. Vi kan skrive en kommentar på en enkelt linje, eller over flere linjer.

Vi skriver enkeltlinjekommentarer med «#». Koden nedenfor viser et eksempel hvor vi skriver kommentaren *over* koden vi ønsker å beskrive.

```
#print("kommentar")
print("ikke kommentar")
```

I programmet ovenfor vil kun «ikke kommentar» bli skrevet ut.

Vi kan også skrive kommentarene på slutten av kodelinjen. Slik:

```
print("ikke kommentar") #kommentar på slutten av linjen.
```

Vi kan skrive mange linjer med kommentar ved å bruke """ for å signalisere starten av kommentaren og en ny """ for å signalisere slutten av kommentaren. Se kodeeksempelet under.

```
"""
starten på mange linjer med kommentar
Alt jeg skriver her er kommentert ut.
Til og med når jeg skifter linje.
Avslutter mange linje kommentaren med en ny tripell apostrof
"""
```

Heldigvis fargelegger de fleste tekst-editorer kommentarer med en tydelig farge, for å skille mellom kode og kommentarer.

Oppgave 27

Lag passende kommentarer til programmene i forrige del.

Oppgave 28

Et problem med kommentarer i koding er at man som ny programmerer ofte skriver for mange kommentarer.

Se på programmene du nettopp har skrevet kommentarer til. Er det noen av kommentarene som er selvforklarende?

Oppsummering

INTRO

Lage variabler

```
tekst = "Hei"
heltall = 2
desimaltall = 0.123
```

Regler variabelnavn:
Ingen spesialtegn som ,.-§!+ eller lignende

1. Ingen tall i starten av variabelnavnet
2. Ingen ord som blir brukt av Python. Disse ordene blir farget når du skriver dem i trinket.
    - Eksempler: print, if, else, for, while, min, max

Pluss, minus, gange, dele

```
pluss  = 10+5
minus  = 10-5
gange  = 10*5
dele   = 10/5
```

NB: tekst kan ganges med heltall.

```
"ERIK"*5
```

Skriv ut tekst til resultatvinduet

```
print("tekst")
```

Skriv ut en variabel til resultatvinduet

```
variabel = 2
print(variabel)
```

Skriv ut flere ting på en gang (bruk komma)

```
print(2,3,4,"hei")
```

Feilmeldinger inneholder et linjenummer som forteller deg hvilke linje feilen er på.

EXTRA

Lage flere variabler på en linje

```
a,b,c = 1,2,3
```

Input fra bruker. NB: Alt som brukeren skriver vil bli lest som tekst av dataen.

```
brukerInput = input("input:")
```

Forandre en variabel til heltall

```
variabel = input("Input:")
variabel = int(variabel)
```

Forandre en variabel til desimaltall

```
variabel = input("Input:")
variabel = float(variabel)
```

Forandre en variabel til tekst

```
variabel = input("Input:")
variabel = float(variabel)
variabel = str(variabel)   #<--
```

Lage enkelt kommentarer i koden

```
# "#" starter en kommentar
variabel = "ikke kommentar"
```

Lag kommentar over flere linjer

```
"""
trippel " starter en
fler linjet kommentar
"""
```

Intro til logikk i programmering

Påstander, verdier og operasjoner – Hva er forskjellen?

Dersom du skal lage mer komplekse programmer er det viktig at du forstår begrepene og ideene **påstander**, **verdier** og **operasjoner**.

Hva betyr disse tre begrepene? La oss se på noen eksempler:

1. **Påstand**

   En **påstand** er noe som er rett eller galt. Hvis noen sier «Det er sol i dag» så er det en **påstand**, fordi det kan være rett eller galt. I Python bruker vi begrepene True eller False, på norsk sant eller usant.

2. **Verdi**

   Hva er en **verdi**? Verdier er noe. Altså, 100 kroner, 18 år eller «Erik». Verdier er verken sanne eller usanne, men sier noe om hva du har.

3. **Operasjon**

   En **operasjon** er noe som skal gjøres. Du kan gå til butikken eller PC'en kan skrive ut en variabel. Om du har skrevet ut en variabel eller om du har gått til butikken kan være sant eller usant, men gå til butikken eller skriv ut en variabel er noe å gjøre, altså en operasjon.

Påstand	Verdi	Operasjon
Sann/usann	Noe	Noe å gjøre

Se på setningene nedenfor. Hvilke er påstander, verdier eller operasjoner?

- Klokken er 12.
- Det er fredag.
- Du er en gutt.

- Klokken 12.
- En gutt og en jente.
- Solen.

- Løp en tur.
- Snakk med læreren.
- Vær forsiktig

Dette blir veldig viktig når du begynner på den neste delen av arbeidsheftet. Før du kan gå videre skal du jobbe med påstander og operasjoner, og begrepene **hvis**, **så lenge** og **for** (engelsk: **if, while** og **for**).

Spørsmål til oppgaver

Under står det mange spørsmål. Oversett dem til påstander og bestem deg for om påstandene er sanne eller usanne for deg. Vi skal bruke dem til oppgavene under.

1. har du vært i 2 land?
2. har du vært i 3 land?
3. har du vært i 4 land?
4. drømmer du om en rosa bil?
5. drømmer du om å eie en BMW?
6. spiller du playstation?
7. spiller du xbox?
8. legger du deg før klokken 22 om kvelden?
9. legger du deg før klokken 03 om kvelden?
10. leser du nyhetene på NRK.no?
11. leser du nyhetene på VG.no?
12. kan du snakke 3 språk?
13. kan du snakke mer enn 3 språk?
14. spiller du PubG på skolen?
15. har du på deg sokker?
16. har du på deg genser?
17. syklet du til skolen i dag?
18. gikk du til skolen i dag?
19. tok du t-banen til skolen i dag?
20. lager du din egen matpakke?
21. Løper du raskere enn læreren din?
22. Er du over 16 år?
23. Er du over 20 år?
24. Er du over 25 år?

Oppgavene i dette kapittelet passer godt til en felles gjennomgang. Om du sitter alene og jobber med oppgavene, så kan du skrive hva du skulle ha gjort, men det er ingenting som stopper deg fra å gjøre det oppgaven spør om.

Oppgave 29

Punkt 9 er legger du deg før klokken 03 om morgen.
Hvis det stemmer for deg, så skal du klappe med hendene.

Oppgave 30

Punkt 1 er om du har vært i 2 land.
Hvis du har vært i to land, så skal du klappe med hendene.

Oppgave 31

Hvis punkt 3 stemmer for deg, så skal du klappe med hendene.

Oppgave 32

Hvis punkt 22 og punkt 10 stemmer for deg, så skal du reise deg.

Oppgave 33

Hvis punkt 11 stemmer for deg, så skal du reise deg.

Oppgave 34

Hvis punkt 15, så reis deg og klapp med hendene.

Oppgave 35

Hvis = if

if punkt 16, så skal du si Ja.

Oppgave 36

Siste steg. Hvordan tror du at du skal lese det som står under?

```
if punkt8:
    klappMedHendene()
```

Oppsummert: h**vis** betyr at du skal sjekke om spørsmålet/påstanden stemmer for deg. **If** er det engelske ordet for hvis og det ordet vi bruker i Python for å få Python til å sjekke en **påstand**. Dersom påstanden stemmer må du gjøre operasjonen som følger. Hvis den ikke stemmer skal du gjøre ingenting.

Hvis punkt 9 stemmer for deg, så skal du klappe med hendene.

```
If punkt9:
    klappeMedHendene()
```

En **påstand** kan kun ha to utfall; True eller False **(sant** eller **usant)**. Tidligere i heftet leste du at på programmeringsspråket blir:

- heltall kalt «**int**»
- desimaltall kalt «**float**»
- tekst kalt «**string**».

På samme måte blir True og False kalt «**boolean**». En **boolean** er en variabel som kun kan ha to utfall (True eller False, ja eller nei, 0 eller 1, av eller på, osv.). Fagordet for **påstand** er **boolske uttrykk**, siden en **påstand** bare kan ha to mulige utfall. Dersom du noen gang leser eller hører om begrepet «**boolske variabler**», betyr dette en **variabel** som bare kan ha to mulige verdier.

Dersom vi skriver en **påstand** vil Python anse det som en **boolean**, fordi utfallet må være enten True eller False. Vi kan også lage en **boolean** ved å skrive:

Variabelnavn = True eller False

For eksempel:

```
punkt9 = True
punkt11 = False
```

Tall og sammenlignende operatorer

I realfagene kan vi kan vi ha behov for å sjekke om verdier er like eller ulike hverandre. Nedenfor ser du en liste over **boolske operatorer**, og hva de betyr:

Operator	**<**	**<=**	**>**	**>=**	**==**	**!=**
Betydning	Mindre enn	Mindre enn eller like	Større enn	Større enn eller like	like	Ikke like
Eksempel på en påstand	2 er mindre enn 3	4 er mindre eller lik 3	5 er større enn 4	5 er større eller lik 6	6 er lik 6	7 er ikke lik 7
Kode eksempel	`2 < 3`	`4 <= 3`	`5 > 4`	`5 >= 6`	`6 == 6`	`7 != 7`
Resultat	True	False	True	False	True	False

Oppgave 37

Hvilke verdi vil variabelene under ha?

```
påstand = 2>0      uttrykk = 10*10 > 10      variabel = True
navn = False      stortTall = 10**10 < 1     liteTall = 1 < 10**10
```

Oppgave 38

Per vil sjekke om alderen sin er under 3^3, men han sjekket på kalkulatoren at det ble 27 og han er ikke over 27, men PC'en sier True. Kan du finne feilen?

```
alder = 17
print(alder > 3*3)
```

Oppgave 39

Oda vil ikke at person1 og person2 skal være samme alder. Programmet bare krasjer, hva er feil? Hint: det er to feil

```
person1 = 18
person2 = 17
print(person1 ! person2)
```

Oppgave 40

Lag en variabel som lagrer om du er eldre enn 2^4.

Oppgave 41

Erik har lyst til å sjekke påstanden «Per og Pål», men hva er feil med det Erik tenker?

Oppgave 42

Vi skal lage et program som sjekker om du og jeg er eldre eller lik 42 år gammel. Jeg er 26 år gammel. Skriv programmet som gjør dette.

Oppgave 43

Fullfør programmet slik at det skriver ut om du er eldre enn 16, 17 og 18.

```
Alder16 = 16
Alder17 = 17
Alder18 = 18
dinAlder = __
print(
print(
print(
```

Tekst og sammenlignende operatorer

Av og til så finnes det eksempler hvor det å sammenligne tekst er nyttig, men da må vi ha klart for oss hva det er vi skal sammenligne. Teksten "Erik" og teksten "Per" kan sammenlignes på to forskjellige og nyttige måter. Vi kan enten sammenligne hvilke av ord som kommer først i alfabetet, eller vi kan sammenligne hvilke av ordene som er lengst. Her tenker du kanskje «ja men jeg kan jo bare sjekke selv». Men, hva hvis du skulle sjekket 2 000 navn eller ord? Du ville brukt kjempelang tid og blitt veldig lei, men PC'en din ville gjort det uten å mukke på bare noen sekunder. Så den egentlige verdien av sånne enkle programmer blir først tydelig når vi har store mengder med data.

Når vi bruker det boolske operatorene med vanlig tekst, sjekker Python alfabetisk etter en sammenheng, men om du vil sjekke etter hvilke ord som har flest tegn, så kan vi bruke len() funksjonen rundt teksten vår.

```
len("Erik") > len("Per") #Erik har flere bokstaver enn Per  - True
"Erik" > "Per"         #Erik er senere alfabetisk enn Per    - False
```

Når vi sammenligner med len-funksjonen, så sjekker vi lengden av ordet. Prøv koden, print(len("DittNavn")). Koden vil skrive ut tallet 8, siden det er 8 bokstaver i "dittnavn". Ordet «Erik» består av fire bokstaver og er dermed lengre enn ordet "Per" som består av 3 bokstaver. Derimot, om vi sammenligner om "Erik" er større enn "Per", så sjekker vi hvilke av ordene som kommer først alfabetisk. Erik starter med E og er dermed tidligere i alfabetet og tolkes derfor som mindre enn "Per".

Oppgave 44

Hva gjør programmet nedenfor?

```
navn = "Programmering"
lengdeNavn = len(navn)
print(lengdeNavn,navn)
```

Oppgave 45

Hva blir skrevet ut fra programmet?

```
mat = "Matematikk"
nat = "Naturfag"
print(mat>nat)
print(len(mat)>len(nat))
```

Oppgave 46

Skriv en kode som sier hvor langt verdens lengste personnavn er.

Hubert Blaine Wolfeschlegelsteinhausenbergerdorff Sr.

Oppgave 47

En landsby i Wales har rekorden for lengste stedsnavn. Skriv et program som skriver ut om verdens lengste personnavn er lengre enn verdens lengste stedsnavn.

Llanfairpwllgwyngyllgogerychwyrndrobwllllantysiliogogogoch

Oppgave 48

Gjør de to forrige oppgavene, men med ditt navn og din adresse.

Oppgave 49

Hvilke av påstandene er True?

```
var1 = "Test"
var2 = "test"
var3 = "variabel"
var4 = "operator"
var5 = "påstander"

var1 == var2         var2 >= var1      var1 > var4
len(var5) > var4      var4 != var3      len(var5) > len(var3)
len(var2) != len(var1)  len(var1) == 4     var1 > var5
```

Oppgave 50

I den forrige oppgaven var det oppgitt 5 forskjellige variabler. Legg til "aaa" først i alle tekstene, for eksempel var1 blir "aaaTest". Hvilke av påstandene i forrige oppgave er nå sanne?

Oppsummering

Påstand, verdi og operasjon

Du har 100 kr, 100 kr, skaff deg 100 kr

Påstand	Verdi	Operasjon
Sann/usann	Noe	Noe å gjøre

Oversikt over Sammenlignende operatorer i Python

Operator	**<**	**<=**	**>**	**>=**	**==**	**!=**
Betydning	Mindre enn	Mindre enn eller like	Større enn	Større enn eller like	like	Ikke like
Eksempel på en påstand	2 er mindre enn 3	4 er mindre eller lik 3	5 er større enn 4	5 er større eller lik 6	6 er lik 6	7 er ikke lik 7
Kode eksempel	2 < 3	4 <= 3	5 > 4	5 >= 6	6 == 6	7 != 7
Resultat	True	False	True	False	True	False

Lagre påstander i Python

```
påstand1 = True
påstand2 = False
påstand3 = 3 > 2.5              #Tallverdi                    - True
påstand4 = "Erik" > "Per"       #Alfabetisk rekkefølge - False
påstand5 = len("Erik") > len("Per") #Lengden av ordene        - True
```

If – elif - else

If-test

I forrige kapittel fikk du en introduksjon til hva **påstander** er og hva ordet «**if**» betyr. Videre leste du om hvordan du kan lage **påstander**. Et eksempel på hvordan du skriver if-tester ble presentert slik:

> Hvis punkt 9 stemmer for deg, så skal du klappe med hendene.
>
> If punkt9:
> klappeMedHendene()

PC'en leser eksempelet som «hvis påstand 9 er true, så skal jeg gjøre funksjonen klappeMedHendene()». Nedenfor finner du et mer fullverdig eksempel som du kan teste i Trinket.io. Vi lager to variabler med et tall hver. Deretter skal vi sjekke hvilke som er størst:

```
minAlder = 26
dinAlder = 16
if dinAlder > minAlder:
    print(dinAlder)
```

I eksempelet over, så er ikke dinAlder større enn minAlder, altså False, dermed skriver vi ikke ut dinAlder. Vi ville at den største alderen skulle bli skrevet ut, så la oss lage enda en if-test som sjekker motsatt vei.

```
minAlder = 26
dinAlder = 16
if dinAlder > minAlder:
    print(dinAlder)
if minAlder > dinAlder:
    print(minAlder)
```

I eksempelet over, så sjekker vi først om dinAlder er større enn minAlder og hvis det stemmer, så skriver vi ut dinAlder. Uavhengig av hva som er svaret på den første **if-testen**, så sjekker vi etterpå om minAlder er større enn dinAlder.

Her er det imidlertid et lite problem:

Tenk deg at dinAlder er 26 og minAlder er 16, altså motsatt av forrige eksempel. Hvis Python sjekker om «dinAlder er større enn minAlder» så stemmer det, og Python skriver ut dinAlder.

Imidlertid vil programmet fortsatt sjekke om minAlder er større enn dinAlder, noe som er helt unødvendig. Vi vet jo allerede at det ikke er sant, fordi dinAlder var større enn minAlder. Dette fremstår kanskje som en bagatell, men det kan få store konsekvenser når programmene blir større og mer komplekse. Jo flere operasjoner en PC skal utføre, jo lengre tid bruker PC'en på å utføre alle operasjonene. Dermed er det fornuftig å unngå unødvendige operasjoner når vi skriver koder.

Elif-test

For å løse dette «problemet» kan vi bruke funksjonen elif istedenfor if foran den andre **påstanden**. Da sjekker Python om den andre **påstanden** stemmer kun dersom den første *ikke* stemmer.

Vi tar utgangspunkt i programmet på forrige side, men ønsker at Python kun sjekker om «minAlder er større enn dinAlder» hvis det ikke var sant at «dinAlder er større enn minAlder», altså hvis den første påstanden var False.

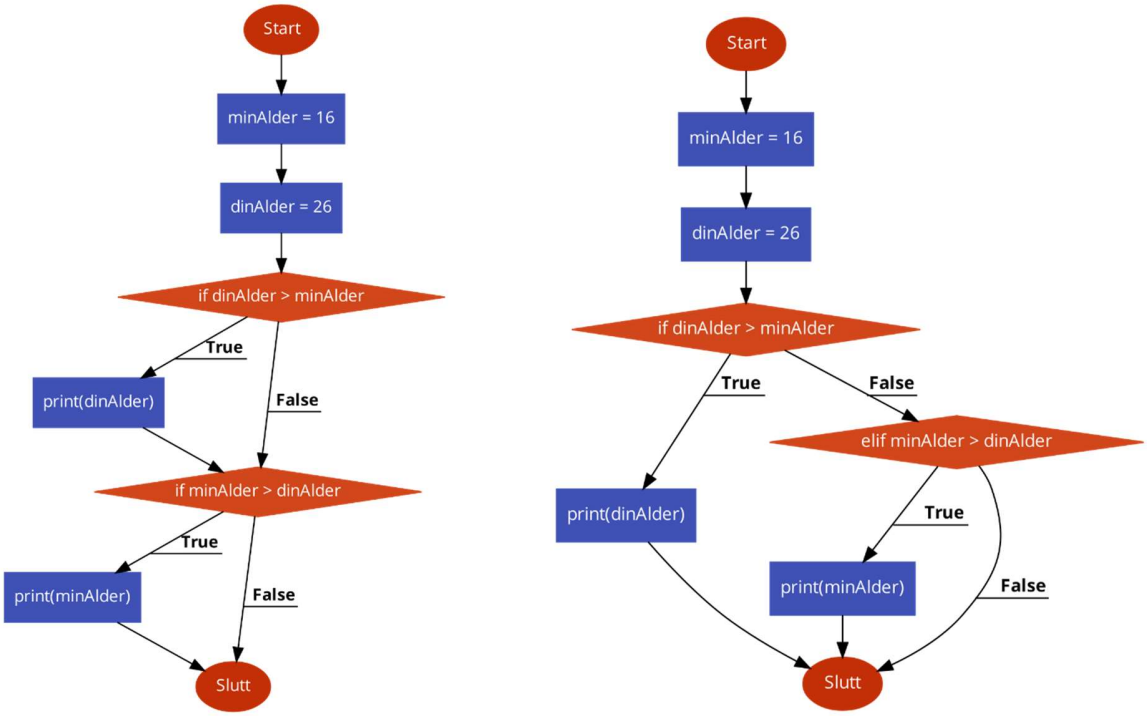
Prøv å kjøre koden nedenfor, og forklar den til en klassekamerat.

```
minAlder = 16
dinAlder = 26
if dinAlder > minAlder:
    print(dinAlder)
elif minAlder > dinAlder:
    print(minAlder)
```

Bildene på neste side kalles **flytskjema** («**flowchart**» på engelsk). De kan være nyttige å lage for å visualisere hvordan Python kjører de forskjellige kodebitene som er vist til nå.

Du leser et flytdiagram med å starte på Start-feltet, og følger pilene til du kommer til Slutt-feltet. Dersom du kommer til en **påstand**, må du selv avgjøre om **påstanden** er True eller False.

Flytskjema



Flytskjemaet på venstre side viser hvordan Python kjører to **if-tester** etter hverandre, mens **flytskjemaet** på høyre side viser hvordan en **if- test** etterfulgt av en **elif-test** blir kjørt i Python.

Else

Tenk deg at vi kun er interessert i å sjekke om **påstanden** «dinAlder er større enn minAlder» er True. Dersom **påstanden** er True, ønsker vi at Python skal printe «dinAlder». Dersom den er False, ønsker vi *ikke* at Python skal sjekke om «minAlder er større enn dinAlder» eller om vi er like gamle.

Dersom vi ønsker dette, kan vi skrive koden slik:

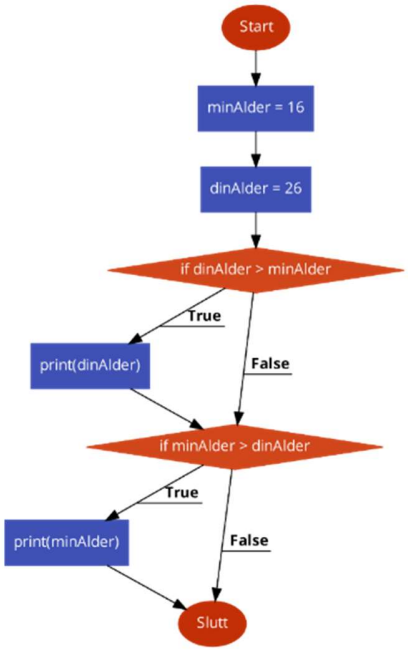
Hvis dinAlder er større enn minAlder, så skal Python printe «dinAlder»

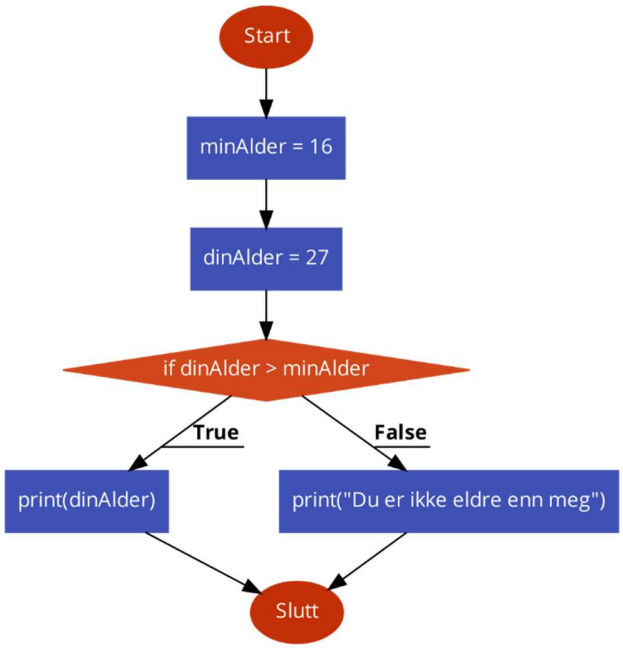
Hvis ikke, så skal Python printe noe annet, for eksempel en tekst som «Du er ikke eldre enn meg».

Det engelske ordet for «**hvis ikke**» er «else». Vi bruker else for å si til PC'en at den ikke trenger å sjekke noe, men gjøre det som er skrevet etter else dersom **if-testen** *ikke* var True. Kjør programmet til høyre, og sjekk om du forstår det.

```
minAlder = 26
dinAlder = 16
if dinAlder > minAlder:
    print(dinAlder)
else:
    print("Du er ikke eldre enn meg")
```

Flytskjemaet til venstre viser gangen i det opprinnelige programmet, mens **flytskjemaet** til høyre viser operasjonene Python utfører dersom vi bruker else.





Oppgave 51

Hva mangler i koden under? Kjør koden på trinket.io for å sjekke om du har rett.

```
alder = 10
alder2 = 20

if alder+alder2 > 20_
   print("Vi er eldre enn 20 år_
```

Oppgave 52

Hva er forskjellen på en if-test og en elif-test?

Oppgave 53

Odin vil sjekke om han har råd til 5 colaer når en cola koster 17 kroner og han har 90 kroner. If-testen hans sier at han har råd, men Odin har sjekket på kalkulatoren og vet at han ikke har råd. Hva er feil?

```
colaPris = 17+2
if colaPris*5 > 90:
   print("Du kan kjøpe 5 colaer")
```

Oppgave 54

Espen vil skrive en if-test for å sjekke om han har lengre navn enn Ola. Skriv if-testen for Espen.

```
ola = "Ola Nordmann"
espen = "Espen Askeladden"
```

Oppgave 55

Forklar koden under:

```
if "Quebec" > "Paris":
   print("Quebec")
else:
   print("Paris")
```

Oppgave 56

Skriv en kode som sjekker om du kan kjøpe en TV til 3995 kr og 13 Filmer til 135 kr per stykk når du har 6000 kroner. Hvis du kan kjøpe produktene, så må du skrive ut hvor mye penger du har igjen etter kjøpet.

Utfordringer:

Oppgave 57

Lag en variabel som heter inntekt som sier hvor mye en person tjener. Lag en if-test som sjekker hvor mye personen må betale i skatt. Under står det tre skattenivåer. Skriv ut hvor mye personen må skatte. Hint: Du må lage en if-/elif-test for hver av skattenivåene.

30% skatt om man tjener mindre enn 500'000
40% skatt om man tjener mer enn 500'000 og mindre enn 1'000'000
50% skatt om man tjener mer enn 1'000'000

Eksempel:
inntekt = 400000

Utskrift: Med en lønn på 400000 må du betale 120000 i skatt.

Oppgave 58

Lag et program som bestemmer hvem som vinner av to spillere i stein, saks og papir. Hint: Lag to variabler. En for hver spiller. Lagre "Stein", "Saks" eller "Papir" og bruk if-tester for å sjekke hva de har valgt og bestemme vinneren.

Oppgave 59

Lag et program som skriver ut tre navn i alfabetisk rekkefølge, uavhengig av hvilke av navnene som er variabel en, to eller tre.

Eksempel:
navn1 = "Stein"
navn2 = "Espen"
navn3 = "Henrik"

---If-tester---

Svar:

Espen, Henrik, Stein

Oppsummering

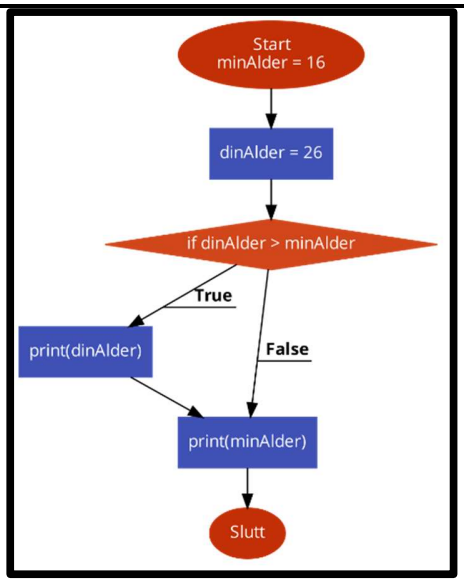
Hvordan lese if-tester i Python

Hvis 2 er større enn 1, så skal du skrive ut 2.

```
If 2>1:
    print(2)
```

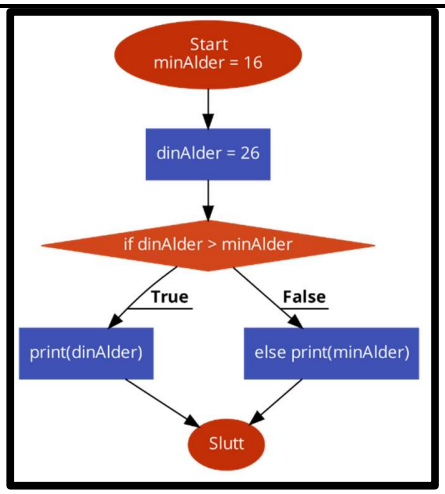
If-test

```
minAlder = 16
dinAlder = 26
if dinAlder > minAlder:
    print(dinAlder)
print(minAlder)
```



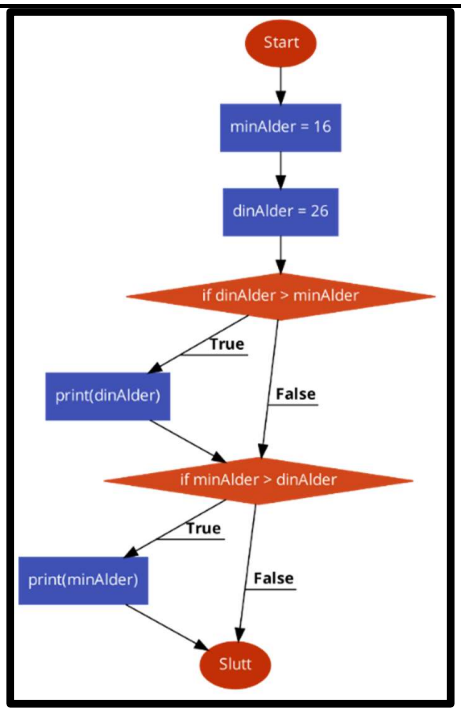
If-Else-test

```
minAlder = 16
dinAlder = 26
if dinAlder > minAlder:
    print(dinAlder)
else:
    print(minAlder)
```



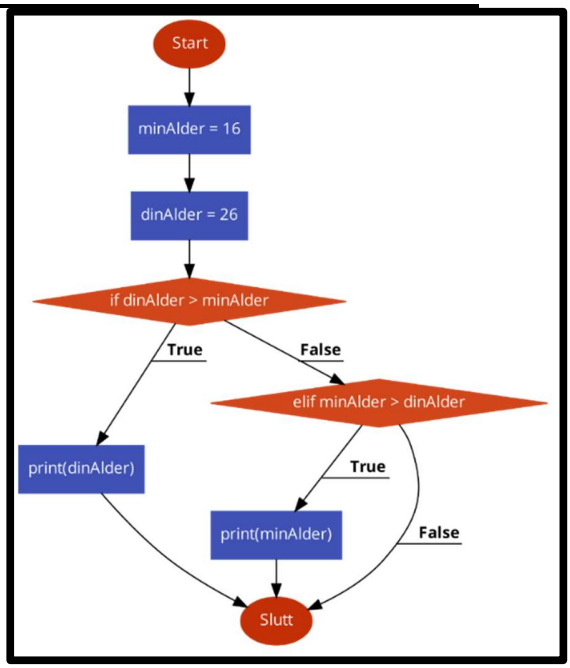
If-If-test

```
minAlder = 16
dinAlder = 26
if dinAlder > minAlder:
    print(dinAlder)
if minAlder > dinAlder:
    print(minAlder)
```



If-Elif-test

```
minAlder = 16
dinAlder = 26
if dinAlder > minAlder:
    print(dinAlder)
elif minAlder > dinAlder:
    print(minAlder)
```



Løkker

Det kan hende at vi ønsker at en kode skal kjøres **så lenge** en **påstand** er sann, eller at koden skal kjøres et **bestemt antall ganger**.

I så fall må vi skrive koden på en måte som kalles **løkke**, som betyr at vi ber Python utføre en eller flere operasjoner inntil et mål er oppnådd. Vi må altså skrive koden på en slik måte at Python vet når programmet skal avsluttes; hvis ikke vil Python kjøre programmet i en evig **løkke**, og PC'en vil til slutt krasje.

While-løkker

Det engelske begrepet for «**så lenge**» er «while», og i dette delkapittelet skal du bli kjent med hva en **while-løkke** er.

Tidligere har vi lært at det som står inne i en if-test, kun blir kjørt om if-testen er sann. Se eksempelet under.

```
                        if "Quebec" > "Paris":
koden inni if-testen  →       print("Quebec")
```

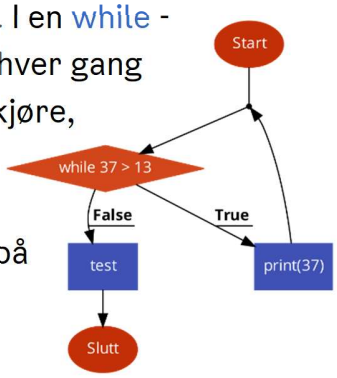
Legg merke til at koden over kun blir kjørt en gang. Dersom vi ønsker å få utskriften "Quebec" flere ganger, kan vi gjøre dette ved å bruke while.

Det kan hende at vi flere ganger vil sjekke om noe stemmer, og for hver gang det stemmer skal noe printes.

Se på følgende eksempel:

```
                        while 37 > 13:
koden inni if-testen  →       print(37)
```

I koden over sjekker Python om 37 er større enn 13. Det er sant, så Python skriver ut 37. Hvis det hadde vært en if -test, hadde vi vært ferdig. I en while -løkke sjekker Python **påstanden** helt den ikke lenger er sann. For hver gang Python sjekker, skriver Python ut 37. Koden over vil aldri slutte å kjøre, fordi **påstanden** alltid er sann. Prøver du programmet på Trinket.io, så slutter trinket å kjøre programmet etter en stund. Om du kjører koden på egen PC, så vil PC din bruke all kraften sin på dette programmet, bli veldig treig helt til du manuelt avslutter programmet. Til høyre ser du et flytdiagram for programmet.

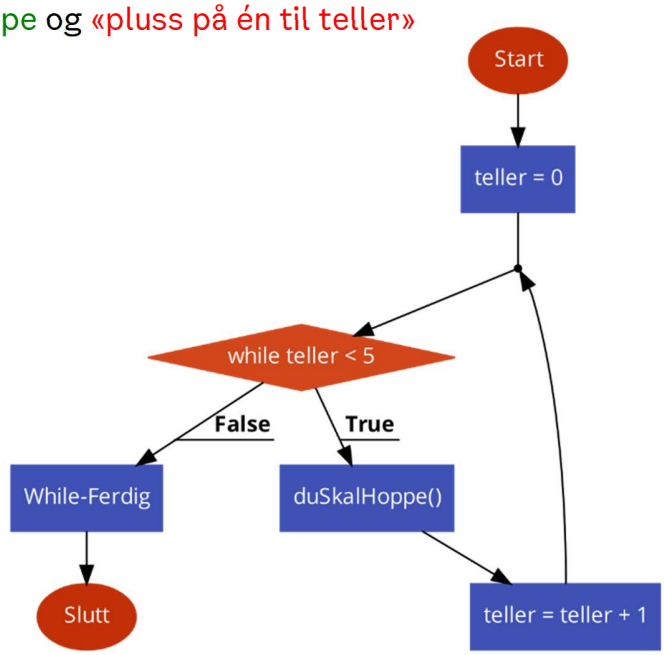


Trikset for at while -løkken skal stoppe er å lage en påstand som forandrer seg. Tenkt deg at læreren din sier at hen skal telle *til* 5, og for hvert tall læreren sier så skal du hoppe en gang.

Læreren starter å telle, og sier tallene 0, 1, 2, 3, 4. Da skal du hoppe en gang for hvert av de grønne tallene (tallene under 5). I en kode vil dette se slik ut:

Mens teller er mindre enn 5, så skal du hoppe og «pluss på én til teller»

```
teller = 0
while teller < 5:
    duSkalHoppe()
    teller = teller + 1
```



Legg merke til at læreren ikke sier 5. Ønsker vi at 5 skal være med, må vi skrive «til og med 5».

Så lenge man har en påstand som forandrer seg over tid og til slutt blir False, så krasjer ikke programmet ditt. Det er flere måter å sørge for at påstanden blir False. I neste del av hefte blir det vist en ny måte å gjøre dette på, men det er ikke viktig for alle å kunne dette. Du kan fint hoppe over denne delen av kapittelet.

Oppgave 60

Pål har skrevet en while-løkke som skal skrive ut tallene 1, 2, 3, 4, 5, 6, 7, 8, 9 og 10. Hva mangler Pål for at programmet skal fungere?

```
teller = 0
while teller < 10:
teller = teller + 1
print(teller)
```

Oppgave 61

Dina vil skrive ut tallene 2, 4, 6 og 8, men programmet skriver ikke ut noe som helst. Hva er feil?

```
teller = 0
while teller > 10:
  teller = teller + 2
  print(teller)
```

Oppgave 62

Hva gjør koden nedenfor?

```
teller = 1
while teller < 10:
  print("*"*teller)
  teller = teller + 1
```

Oppgave 63

Hva gjør koden nedenfor?

```
teller = 1
while teller < 10:
  print(" "*(9-teller),"*"*teller)
  teller = teller + 1
```

Oppgave 64

Hva gjør koden nedenfor?

```
teller = 1
while teller < 10:
  print(" "*teller, "*"*(19-(2*teller)), " "*teller)
  teller = teller + 1
```

Oppgave 65

Skriv en while-løkke som printer ut tallene 5,4,3,2,1 og 0 i den rekkefølgen.

Oppgave 66

Skriv et program som sjekker om du er over 25. Hvis du ikke er over 25, så skal du bli et år eldre helt til du er 25.

Oppgave 67

Utvid programmet fra forrige oppgave slik at det teller hvor mange år det tok før du var 25 år gammel. La programmet skrive ut denne telleren etter while-løkken.

NB: dette programmet kan du bruke senere i 1P.

Oppgave 68

Per vil lage et program som regner ut summen av alle tallene fra 1 til 10. Han har skrevet programmet under. Fiks programmet så det gjør det Per vil at det skal gjøre.

```
summen = 0
teller =
while teller < 10
  teller += 1
  summen += teller
print(summen)
```

Oppgave 69

Lag et program som regner summen for alle oddetall opp til tallet 100. Gjør det samme for partallene. Kan man gjøre dette med en while-løkke alene eller må man bruke if-tester eller to while-løkker?

Nyttig, men ikke viktig informasjon om while-løkker

Du kan si påstand1 = True og si while påstand1, for deretter å skifte påstand1 inne i while-løkken. Koden nedenfor viser dette:

```
påstand1 = True
while påstand1:
    påstand1 = False
```

Koden over vil i prinsipp fungere som en if-test. Påstand1 er True, nøyaktig en gang. Etter første kjøring igjennom koden inne i while-løkken, så er påstand1 satt til False. Dette kan vi utvide til å bruke en if-test for å sjekke om vi vil slutte. I koden under sjekker vi om verdien til variabelen teller opphøyd i 3 er større enn 30 og hvis ikke, så legger vi på 1 til teller.

```
teller = 0
påstand1 = True
while påstand1:
    if teller**3 > 30:
        påstand1 = False
        print(teller**3)
    teller = teller + 1
```

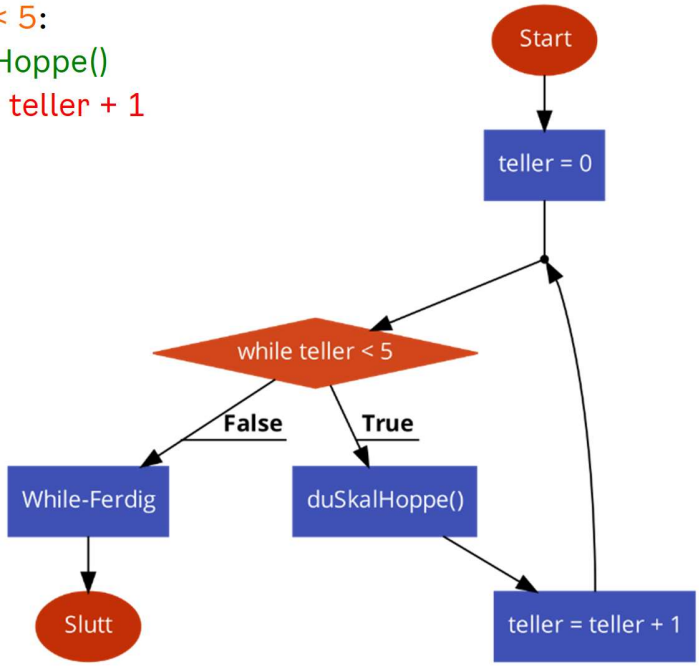
Dette er bare for å vise et eksempel om hvordan vi kan lage en while-løkke, som i utgangspunktet går evig, men som vi likevel avslutter på et tidspunkt. Du kan bruke denne teknikken til å lage mer avanserte while-løkker, men vit at om du gjør dette, så kan du som oftest fikse problemet på en raskere måte. Koden over kan skrives om til koden under. Kjør koden om du vil se resultatet.

```
teller = 0
while teller**3 < 30:
    teller = teller + 1
print(teller**3)
```

Oppsummering

Mens teller er mindre enn 5, så skal du hoppe og «pluss på én til teller»

```
teller = 0
while teller < 5:
    duSkalHoppe()
    teller = teller + 1
```



Tips til while-løkker:

1. Påstanden må være True når du starter while-løkken for at koden inni while-løkken skal bli kjørt.
2. Sørg for at påstanden blir False senere i programmet, hvis ikke risikerer du å krasje PC'en din.
3. Ta utgangspunkt i eksempel-koden under når du lager en while-løkke.

```
teller = 0
sluttpunkt = 10
while teller < sluttpunkt:
  #skriv koden du vil kjøre for hver teller
  teller+=1
```

Lister

En måte å instruere Python til å gjøre en operasjon et bestemt antall ganger, er å bruke en for-løkke.

Før du kan lære hva en for -løkke er, må vi introdusere en ny type variabel; **lister**. **Lister** er omtrent sånn som du sikkert tenker. For eksempel kan en **liste** med navn skrives som dette: Per, Pål og Espen. Det første elementet i **lista** er «Per», og det siste elementet i lista er «Espen».

I Python skriver vi **lister** med klammeparenteser [...], og vi bruker komma mellom alle elementene. For å lage klammeparenteser trykker du AltGr+8 og 9.

Lista [Per, Pål og Espen] skrevet i Python er vist nedenfor:

```
liste1 = ["Per","Pål","Espen"]
#for å skrive '[ ]', så trykker du «Alt Gr + 8»
#Alt Gr knappen er på høyre siden av mellomromsknappen
```

Alle verdiene, det vi kaller element når verdien er i en **liste**, får en plassering. Vi kaller en plassering for en indeks. Akkurat som at vi gir Per indeks «første», så gir Python "Per" indeksen 0. Neste element, "Pål", ville vi sagt er den neste eller på den andre plassen, mens Python sier indeks 1. Slik:

Indeks	0	1	2
Element	Per	Pål	Espen

For å hente ut en verdi i **lista** kan vi skrive variabelnavnet til listen etterfulgt av klammeparenteser med den ønskede indeksen inni klammeparentesen.
liste1[0] vil hente ut det første elementet, altså "Per".

Prøv programmet nedenfor, og prøv å skifte koden slik at du skriver ut "Espen".

```
#indeks   0   1   2
liste1 = ["Per","Pål","Espen"]
print(liste1[0])
```

For-løkker

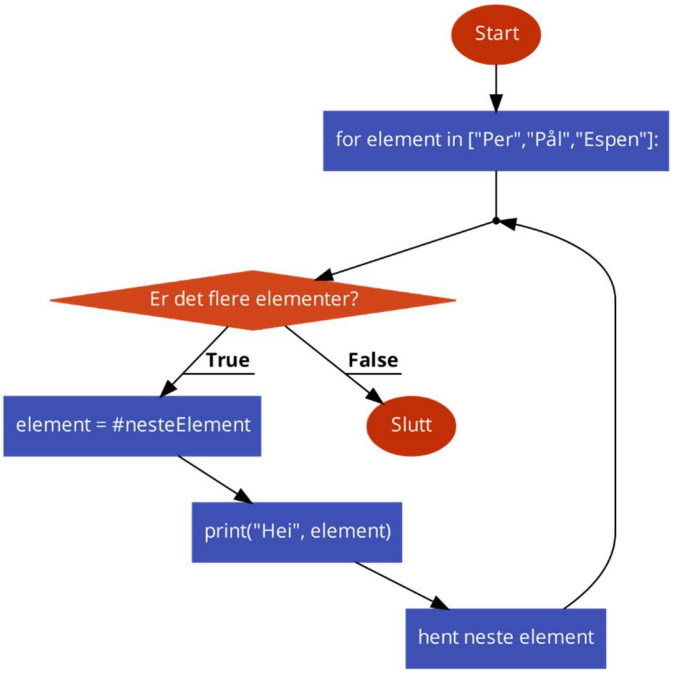
Du har tidligere lært om en while-løkke. Husker du forskjellen mellom en if-test og en while-løkke? If-testen kjørte koden kun en gang, mens while-løkken kjørte koden helt til påstanden var usann, som oftest mange ganger.

Dersom vi ønsker å bestemme hvor mange ganger Python skal kjøre en kode, kan vi bruke en for-løkke. Grunnen til at den kalles en for-løkke er fordi den tar imot en liste og gjør koden for hvert element i lista. Se på eksempelet nedenfor:

```
                        for element in ["Per", "Pål", "Espen"]:
Koden inni for-løkken →     print("Hei", element)
```

Koden over sier at:
for hvert element inni listen, ["Per", "Pål", "Espen"], så skal vi kjøre koden som er inni for-løkken. Utskriften til koden over, kommer til å være «Hei» etterfulgt av første element. Deretter printes det en ny linje hvor det står «Hei» til neste element, og dette fortsetter helt til Python har gjort det for hvert element.

Nedenfor ser du en visualisering av koden i eksempelet over. Sørg for at du forstår visualiseringen før du går videre.



Forskjellen mellom while-løkker og for-løkker

Hva er forskjellen mellom for-løkker og while-løkker? Den viktigste forskjellen er hvordan du ber Python avslutte løkken.

I for-løkker kan du spesifisere hvilke elementer du vil at løkken skal jobbe med. I while-løkker kan du oppgi en betingelse som gjør at den skal slutte. Etter hvert som du jobber med for- og while-løkker vil du forstå når det er best med hvilken av dem.

En bedre måte å skrive for-løkker på

Den forrige måten vi lærte å skrive for-løkke på er veldig nyttig, men litt tungvint når vi skal lage en systematisk for-løkke. Med systematisk mener vi noe som teller på en bestemt måte.

En liste med elementene [0, 1, 2, 3, 4, 5, 6, 7, 8, 9,10] er en liste som teller på en bestemt måte. Elementene i listen starter på 0 og slutter på 10. Hvert tall er én større enn det forrige og én mindre enn det neste.

Utfra denne lista kan vi skrive en kode som printer 10-gangetabellen.

```
for i in [0,1,2,3,4,5,6,7,8,9,10]:
    print(i*10)
```

Legg merke til at vi bruker i som variabelnavn for telleren i for-løkken. Dette er standarden i programmeringsverden. Om du har en for-løkke inne i en annen for-løkke, så bruker vi j for telleren i den andre for-løkken.

Når du skriver en for-løkke som den over, så er det veldig kjedelig og repeterende å skrive inn alle tallene i lista. Tenk om lista skulle inneholdt alle tallene fra 0 til 100 eller 0 til 1000.

Range

Når noe er repeterende i programmering, så kan det sannsynligvis forkortes. Istedenfor å skrive denne listen, [0,1,2,3,4,5,6,7,8,9,10], kan vi bruke range(11). Range er en **funksjon**, akkurat som at print er en **funksjon**, men range(11) lager en *liste*[4].

Hvorfor skriver vi 11? range starter *listen* automatisk på 0 og teller automatisk opp med 1 for hvert tall. range avslutter *listen* på tallet før det du skriver. Om du

[4] Det er ikke en faktisk liste, men et objekt som minner om en liste.

vil at listen skal slutte på 10, så må du skrive 11. De to kodene under gjør nøyaktig det samme.

```
for i in [0,1,2,3,4,5,6,7,8,9,10]:
  print(i*10)
```

```
for i in range(11):
print(i*10)
```

range er faktisk mye smartere enn dette. Det er nemlig tre forskjellige måter å bruke range på. Du kan skrive et, to eller tre verdier inne i range-funksjonen separert med komma «,»:

- Dersom du skriver et tall inni range-funksjonen, vil range automatisk starte på 0 og gå oppover med 1
- Dersom du skriver 2 tall inni range, vil det første tallet være starten og det andre tallet være slutten og range vil automatisk stige med 1
- Dersom du skriver range med tre tall inni, vil første tallet være starten, det andre tallet være slutten, og tredje tallet hvor mye den skal stige med etter hvert tall. Tabellen under prøver å oppsummere dette på en fornuftig måte.

	range(11)	**range(4,11)**	**range(4,11,2)**
Startverdi	Automatisk til 0	4	4
Sluttverdi	10	10	10
Stegverdi	Automatisk til 1	Automatisk til 1	2

Disse tre kodene vil skrive ut de samme tallene. Se om du kan forandre den slik at den midterste bare skriver ut tallene 0,1 og 2 og den siste bare skriver ut 3, 6 og 9.

```
for i in range(5):
  print(i,"et tall")
```

```
for i in range(0,5):
 print(i,"to tall")
```

```
for i in range(0,5,1):
print(i,"tre tall")
```

Oppgave 70

Hva gjør koden under?

```
for i in range(5):
  print(i)
```

Oppgave 71

Håkon vil skrive ut tallene 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 og 10, men Python sier at hun har en feil. Fiks programmet slik at det gjør det Håkon vil at det skal gjøre. Hint: det er to feil.

```
for i in range(0,10)
  print(i)
```

Oppgave 72

Skriv et program som bruker en for-løkke til å skrive ut tallene 5,7 og 9.

Oppgave 73

Gjør det samme som du gjorde i forrige oppgave, men med en while-løkke.

Hvilke av løkkene liker du best?

Oppgave 74

Hva gjør programmet under?

```
liste1 = [2,3,5,7]
for element in liste1:
  print(element)
```

Oppgave 75

Hvordan kunne du brukt range for å lage en løkke som vi ser i forrige oppgave? Kan du lage en while-løkke som gjør det samme?

Oppgave 76

Vi kan bruke koden len() for å finne lengden av listen. Hva tror du skjer når du kjører koden under? Kjør koden etter du har tenkt over hva du tror den gjør.

```
liste1 = [0,1,2,3,4]
tekst = "01234"
print(len(liste1))
print(len(tekst))
```

Fra forrige oppgave kan man se at tekst er en type liste. Hvis vi spør etter lengden til en tekst, så får vi ut antall tegn. Mellomrom vil bli inkludert i utregningen, så "012" vil være like lang som "0 2". Det er to måter vi kan utnytte dette på.

Oppgave 77

Vi har lyst til å skrive ut A,T,C og G etter hverandre. Fullfør koden slik at det skjer.

```
dna = "ATCG"
for index in range(len(dna)):
    print(dna[____])
```

Oppgave 78

I starten av dette kapittelet lærte vi om hvordan vi kan hente ut elementer fra en liste direkte med en for-løkke. Skriv om forrige oppgave slik at du får samme resultat, men uten range-funksjonen.

Utfordringer

Oppgave 79

a) Lag en if-test som sjekker om variabelen "bokstav" er lik "A".
b) En tekst bit er en liste med bokstaver, så vi kan skrive "for bokstav in tekst:" og Python vil bla igjennom teksten, en bokstav av gangen. Skriv et program som går igjennom en tekst og skriver ut hver bokstav på sin egen linje.
c) Bruk det du gjorde i a) og b) til å lage et program som går igjennom teksten og skriver ut bokstaven hvis det er en A.

Hint: Hva skjer hvis du skriver «a» istedenfor «A» i teksten?

Naturfag Oppgave 80

Utvid programmet fra forrige oppgave slik at det sjekker om det står A, T, G eller C. Programmet skal skrive ut T om det er A, A om det er en T, C om det er en G og G om det er en C.

A→T
T→A
G→C
C→G

Test programmet med teksten «ATCG blir TAGC»

Matematikk Oppgave 81

Lag et program som sjekker om tallet er fra og med 0 og til og med 10 og hvis det er det, så skriver programmet ut gangetabellen for dette tallet.

Hint: Du trenger en for-løkke inne i en if-test.

Oppsummering

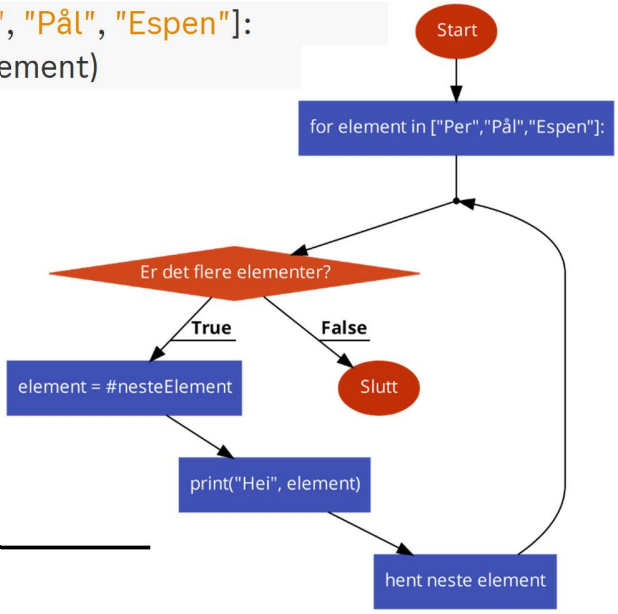
Lister i Python:

```
liste1 = ["Per","Pål","Espen"]
  #for å skrive '[ ]', så trykker du «Alt Gr + 8»
  #Alt Gr knappen er på høyre siden av mellomromsknappen
```

```
          #indeks    0    1     2
          liste1 = ["Per","Pål","Espen"]
          print(liste1[0])
```

For-løkker i Python

```
                              for element in ["Per", "Pål", "Espen"]:
  Koden inni for-løkken →      print("Hei", element)
```



For-løkker med range istedenfor lister

```
for i in [0,1,2,3,4,5,6,7,8,9,10]:        for i in range(11):
  print(i*10)                               print(i*10)
```

	range(11)	**range(2,8)**	**range(4,11,2)**
Startverdi	Automatisk til 0	2	4
Sluttverdi	10	7	10
Stegverdi	Automatisk til 1	Automatisk til 1	2

```
for i in range(5):    for i in range(0,5):   for i in range(0,5,1):
  print(i,"et tall")    print(i,"to tall")    print(i,"tre tall")
```

Ekstra

Funksjoner

Du har kanskje lagt merke til at vi har kalt print, len og range for funksjoner tidligere i heftet, men hva mener vi med det? Vi bruker stort sett ordet funksjoner om matematiske funksjoner, men ordet funksjon betyr egentlig at noe blir gjort. Funksjonen f(x) = 2x, betyr at du skal ta et tall, x, og gange det med 2. Det er noe som skal bli gjort. På samme måte betyr print(tekst) at du skal ta tekst-variabelen og skrive den ut på skjermen.

Hvordan kan vi lage vår egen funksjon? Det er ganske enkelt; vi bruker ordet "def" etterfulgt av hva funksjonen skal hete, samt parenteser som inneholder tingene vi skal bruke i funksjonen. Under ser du et eksempel på dette.

```
def funksjon(test):
    print(test)
```

Her har vi laget en funksjon som heter funksjon, som tar imot en variabel, test, og som skriver ut variabelen til skjermen. Legg merke til at koden som vi vil at funksjonen skal utføre står et innrykk inn for starten av funksjonen. Det er helt likt som for en while-løkke. Nå har vi laget en funksjon, men om du kjører denne koden skjer ingenting. For at Python skal kjøre funksjonen, må vi påkalle funksjonen. Slik:

```
def funksjon(test):
    print(test)

#her påkaller vi funksjonen
funksjon("Skriv ut dette")
```

Hvorfor er funksjoner så nyttige? Funksjoner er nyttige fordi de lar oss bruke en kode flere ganger veldig raskt. De er også nyttig fordi at gode funksjoner gjør programmet mer oversiktlig og forståelig.

Vi bruker oppgave 78 som eksempel. I oppgave 78 skal du lage en tekst, og skrive ut en og en bokstav i teksten. La oss si at du vil lage 5 tekster og skrive ut en og en bokstav for hver av disse tekstene. På neste side ser du hvordan vi må gjøre dette uten funksjoner.

```
tekst1 = "Tekst1"
tekst2 = "Tekst2"
tekst3 = "Tekst3"
tekst4 = "Tekst4"
tekst5 = "Tekst5"
for bokstav in tekst1:
    print(bokstav)
for bokstav in tekst2:
    print(bokstav)
for bokstav in tekst3:
    print(bokstav)
for bokstav in tekst4:
    print(bokstav)
for bokstav in tekst5:
    print(bokstav)
```

Hvis vi bruker funksjoner, kan vi skrive om koden over til å se slik vi har skrevet nedenfor. Dermed ser programmet ryddigere ut og vi har spart noen linjer. Legg merke til at i dette tilfellet har vi bare spart noen linjer, men i et større program kan vi spare mange hundre, om ikke tusenvis av, linjer.

```
tekst1 = "Tekst1"
tekst2 = "Tekst2"
tekst3 = "Tekst3"
tekst4 = "Tekst4"
tekst5 = "Tekst5"
def printBokstaver(tekst):
    for bokstav in tekst:
        print(bokstav)
printBokstaver(tekst1)
printBokstaver(tekst2)
printBokstaver(tekst3)
printBokstaver(tekst4)
printBokstaver(tekst5)
```

Bruk av andres kode

En av styrkene til Python er at man kan bruke kode som andre har skrevet, og at det er et utrolig stort utvalg av kode å hente. Vi kaller andres kode ofte for biblioteker (eng: libraries), eller pakker (eng: packages). Hvorfor og hvordan bruker vi andres kode? Vi bruker andres kode fordi det sparer oss tid og fordi den ofte er bedre enn vår egen kode. Prøv å skrive en kode for å lage en graf som viser linjen y=x mellom x-verdiene 0 og 10. Koden under gjør nettopp dette.

```
from numpy import *
from matplotlib.pyplot import *

plot(linspace(0,10,10))
show()
```

I trinket er det begrenset hvor mange pakker som er tilgjengelig, men om du har en IDE eller bruker terminalen, så kan du ganske lett installere flere. Dette er en av fordelene med å ha Python på egen maskin. Pakkene som er tilgjengelig på trinket er builtins, math, matplotlib.pyplot, numpy, operator, processing, pygal, random, re, string, time, turtle, urllib.request. Av disse er det bare math, random, numpy og matplotlib.pyplot som kommer til å være nyttig for dette arbeidshefte.

Du skal lære to måter å importere andres kode. Den første måten er den vi allerede har sett. Vi skriver «from» etterfulgt av navnet til pakken, og deretter import. Dersom vi skriver «*» betyr at det av vi vil ha alt. Vi kan også spørre etter en bestemt pakke. I eksempelet over ser du at vi bruker plot, linspace og show. Vi kan spesifikt spørre pakkene etter dette.

```
from numpy import linspace
from matplotlib.pyplot import plot,show

plot(linspace(0,10,10))
show()
```

Den andre måten er å skrive «import» etter fulgt av pakkenavnet og så «as» og det du vil kalle pakken. Se eksempel under.

```
import numpy as np
import matplotlib.pyplot as plt

plt.plot(np.linspace(0,10,10))
plt.show()
```

Legg merke til at om du importerer andres kode på denne måten, så må du skrive kallenavnet til pakken før du skriver «.» og navnet til funksjonen du vil bruke. Altså plt.plot, hvor plt er kallenavnet til matplotlib.pyplot og plot er funksjonen du vil bruke. Vår anbefaling er at du bruker:

from «pakkenavn» import *.

Om du lærer deg mer koding, så kommer det fort til å bli et problem, men inntil videre er dette den måten som gjør det best for deg å kode.

Dersom du som elev skal importere pakker, vil dette bli sagt av læreren eller stå i oppgaven.

Komprimerte kode i Python

Python har mange skrivemåter for å gjøre koden mer kompakt og lesbar. Av og til blir den bare mer kompakt, men det kan også være en fordel i seg selv. Vi vil raskt gå igjennom noen komprimerte versjoner av Python kode.

Vi vet at vi kan lagre flere variabler på en linje, som vist under, men vi kan også pakke ut verdier fra en liste på denne måten. Bruker vi * så blir alle verdiene lagret i denne variabelen.

```
liste1 = [1,2,3,4,5]
a, *b = liste1     #a = 1
print(b)         #b = [2,3,4,5]
*c, d = liste1     #d = 1
print(c)         #c = [1,2,3,4]
```

Vi kan bruke ordene and og or i if-tester. Hvis vi bruker ordet and, så må påstand1 og påstand2 være sann, mens hvis vi bruker ordet or, så trenger bare en av påstandene å være sann.

```
if 1==1 and 1==2:      #komprimert versjon if 1==1==2:
 print("success")

if 1==1 or 2==1:
 print("success")
```

If-tester kan man også skrive på en linje. Det ser litt rart ut, men det gjør nøyaktig det samme som om det er over flere linjer.

```
#vanlig versjon
if 2==2:
   print(2)
else:
   print(3)

#kort versjon
print(2) if 2==2 else print(3)
```

Dette gjelder også for for-løkker, men vi må bruke det i kombinasjon med å lage en liste. Nedenfor ser du et eksempel på dette. Resultatene fra for-løkken blir lagret i en liste.

```
test = [i**2 for i in range(11)]
```

Siste komprimeringstrikset vi skal se på i Python er funksjonskomprimering. Det kalles lambda i Python. Istedenfor at vi skriver hele funksjonen i Python, så kan vi skrive det på en linje.

```
def function1(x,y):
   return x+y

function2 = lambda x,y : x+y

print(function1(10,11))
print(function2(10,11))
```

Veien videre

Det du har lært nå er det som er grunnlaget for logisk programmering og fundamentet til det meste av programmering. Det er fortsatt masse å lære om du vil bli flink til å programmere. Enten det er for å lage spill, nettsider eller andre programmer. Det er noen konsepter som er essensiell at du lærer deg. Du har lært helt grunnleggende om lister og bør lære mer om dette før du går videre. Etter det burde du lære om funksjoner og hvordan du bruker andres kode. Til slutt burde du lære om klasser og hvordan du bruker dette med alt annet. Når du har lært disse tingene, så kan du det meste av basis kunnskapen innenfor programmering.

Om du har lyst til å lære disse tingene, så er dette noen ressurser som du kan bruke for å lære deg selv mer, men du trenger ikke å lære noe mer for å jobbe med naturfags og matematikk delen i dette heftet.

freeCodeCamp har det meste av kurs. Alle kursene er gratis og finnes også på YouTube. Kursene er på engelsk. Du kan lære deg alt fra spill utvikling (unity) til hacking via denne nettsiden.

www.freecodecamp.org

https://www.youtube.com/c/Freecodecamp

W3Schools startet som et nettsted for å lære seg kode til WWW, world wide web, men har i senere tid blitt et oppslagsverk for eksempel kode.

https://www.w3schools.com/

Siste nettsiden jeg vil dele er stackoverflow. Om du lurer på noe som helst innenfor programmering, så prøver du å skrive nøkkelordene på engelsk med stackoverflow etter på. Dette er en av de mest brukte nettsidene for å finne svar på programmeringsspørsmål.

https://stackoverflow.com/

Python Kode	Forklaring	Kode - eksempel
=	Er lik tegnet i python er at man definerer venstresiden til å være lik høyresiden. Eksempelet viser at variabel nå skal bety tallet 123	variabel = 123
""	Brukes for å lagre tekst til en variabel.	variabel = "tekst"
+	+ legger sammen verdien på venstresiden med verdien på høyresiden av + tegnet.	variabel = 2 + 3
	I Python kan vi bruke + på tekst også.	variabel = "variabel" + "navn"
-	- trekker fra det på høyre siden fra det på venstre siden.	variabel = 3 – 2
*	* fungerer som multiplisering.	variabel = 3 * 2
/	/ fungerer som deling.	variabel = 3/2
()	() spesifiserer hva som er en del. (2+4)/2 blir 6/2, mens 2+4/2 blir 2+2	variabel = (2+4)/2
**	** blir brukt som opphøyd i. 10^2 kan vi skrive som 10**2 i Python.	variabel = 10**2
#	# er starten på en kommentar. Alt som kommer etter # på den samme linjen er en kommentar	variabel = 2021 #årstall
"""	""" markerer området til en fler linje kommentar.	""" Flere linjer med kommentar """
print()	print brukes for å skrive ut informasjon til resultatvinduet.	print("Erik")
input()	input er for å hente informasjon fra brukeren. Skriv en melding til brukeren	input("navn:") → navn: #bruker skriver

	inni parentesene. Brukeren skriver svaret sitt etter meldingen. Enter for å fullføre.	
int()	int brukes for å sørge for at Python leser objektet som et heltall.	int("2") #heltall → 2
float()	float brukes for å sørge for at Python leser objektet som et desimaltall.	float("2") #desimaltall → 2.0
str()	str brukes for å sørge for at Python leser objektet som tekst.	str(2) #tekst → "2"
type()	type sjekker hvilke type objektet er.	type(2) #int type(2.0) #float type("2") #str → string
==	== sjekker om det på høyre siden er lik det som er på venstre siden	variabel = 2==2#True
<	< sjekker om venstre siden er mindre enn høyre siden.	variabel = 2<3#True
>	> sjekker om venstre siden er større enn høyre siden.	variabel = 2>3#False
<=	<= sjekker om venstre siden er mindre enn høyre siden ELLER om venstre siden er lik høyre siden	variabel = 5<=5#True
>=	>= sjekker om venstre siden er større enn høyre siden ELLER om venstre siden er lik høyre siden	variabel = 5>=5#True
!=	!= sjekker om venstre siden IKKE er lik høyre siden	variabel = 2!=3#True
if	if brukes for å teste om en påstand er riktig/sann eller feil/usann.	if True: print("if-test")

else	Else blir brukt etter en if-test. Det som står i else området vil KUN bli skrevet om if-testen er ikke stemmer/usann.	if False:
		  print("if-test")
		else:
		  print("Dette blir skrevet ut hvis if-testen er False")

elif	Elif er en kombinasjon av else og if. Elif blir brukt etter en if-test. Det som står i elif området vil KUN bli skrevet om if-testen ikke stemmer/er usann OG elif-testen stemmer/er sann.	`if False:`
		`  print("if-test")`
		`elif True:`
		`  print("Dette blir skrevet ut hvis if-testen er False og elif-testen er True")`

len	len må brukes med parenteser og sier hvor mange verdier det er i en list. Tekst er også en list.	`len("if-test")#7` `len([1,2,3])#3`

for	for et kodeord i python som starter en telle-løkke. Løkken trenger å få vite tallverdiene som den skal kjøre igjennom.	`for i in [1,2,3]:`
		`  print(i)`
		`  #skriver ut tallen 1,2 og 3 på hver sin linje`

range	range lager en «liste» ut i fra et til tre tall som brukeren skriver inn. Alle eksemplene lager en «liste» fra og med 0 til 11, ikke med 11.	`range(11)#slutt`
		`range(0,11)#start,slutt`
		`range(0,11,1)#start,slutt,steg`

while	while er en løkke som slutter å kjøre når påstanden blir usann.
	Eksempelet viser hvordan en while løkke kan se ut. Den vil stoppe når alderen er 18 eller over.
		`alder = 16`
		`while alder < 18:`
		`  alder = alder + 1`

Løsningsforslag til programmeringsoppgavene

```
#1
1teller = 123     teller = "riktig"   tekst.2 = "tekst"
!teller = 1       Teller = 1        x = 10
xyz = 2021        alder = 17        navn = 100

#2
"""
1.  Ingen spesialtegn, som ,.-§!+ eller lignende
2.  Ingen tall i starten av variabelnavnet
3.  Ingen ord som blir brukt av Python. Disse ordene blir farget når du skriver dem
i trinket.
o   Eksempler: print, if, else, for, while, min, max
"""

#3
tall1 = 1      tall2 = 2      tall3 = 3

#4
tall1 = 1
tall2 = 2
tall3 = 3

#5
navn = "Erik"
alder = 27

#6
"""
Kaller du variablene for navn og alder, så er det godt nok.
Om du brukte noe som a og b som variabelnavn,
så burde du heller bruke navn og alder.
"""

#7
print("Hello World")
#utskrift --> Hello World
```

```
#8
#mangler --> "
print("Erik")



#9
# mangler --> =
navn = "Erik Skaar"
print(navn)

#10
print("Erik"+"26")
#utskrift --> Erik26

#11
print("Erik "+"26")
#alternativ løsning
print("Erik","26")

#12
alder = 27
navn = "Erik Skaar"
#alternative måter å skrive ut setningen på
print(f"Jeg heter {navn} og er {alder} år gammel")
print("Jeg heter "+navn+" og er "+str(alder)+" år gammel")
print("Jeg heter",navn,"og er",alder,"år gammel")

#13
alder = 26
navn = "Erik"
print(alder*navn)
#utskrift
ErikErikErikErikErikErikErikErikErikErikErikErikErikErikErikErikErikErikErikErikErikEri
kErikErikErikErikErik

#14
#velg en av linjene under
```

```
print("Ola "+"Nordmann")
print("Ola ","Nordmann")
print("Ola Nordmann")

#15
print("Ola "+"Nordmann")#Ola Nordmann
print("Ola ","Nordmann")#Ola  Nordmann
#forskjell --> et ekstra mellomrom

#16
navn = "Erik"
print(5*navn)

#17
tall = 8
print(tall*5)
#utskrift --> 40

#18
tall1 = 7
tall2 = 9
print(tall1+tall2)
print(tall1-tall2)
print(tall1*tall2)
print(tall1/tall2)

#19
tall1 = 8
tall2 = 8
print(tall1*tall2)
print(tall1*(tall2+1))
#mangler () rundt tall2+1

#20
tall1 = 8
tall2 = 9
print(tall1*tall1)
print(tall1*tall2)
```

```
#21
print(10**10)
#mangler --> en *
#10*10 betyr 10 gange 10
#10**10 betyr 10 opphøyd i 10

#22
x = 5
print(10**(x+1))

#23
alder = input("Alderen din er: ")

#24
tall1 = input("tall1:") #brukerinput blir lagret
tall2 = input("tall2:") #ny brukerinput blir lagret
print("sum",tall1+tall2) #skriver ut brukerinputen bak hverandre
#brukerinputen er tekst, IKKE tall, så det er ikke matematisk pluss vi får med +
tegnet
#25
tall1 = int(input("tall1:"))
tall2 = int(input("tall2:"))
print("sum",tall1+tall2)
#mangler --> int(...) hvor ... er input(__)

#26
tall1 = int(input("tall1:"))
tall2 = int(input("tall2:"))
tall3 = int(input("tall3:"))
print("sum",tall1*tall2)
print("sum",tall2+tall3)

#27
"""
Se løsningsforslagene for tidligere oppgaver
her er et eksempel for oppgave 26
"""
```

```
tall1 = int(input("tall1:"))    #brukerinput til tall
tall2 = int(input("tall2:"))    #brukerinput til tall
tall3 = int(input("tall3:"))    #brukerinput til tall
#skriver ut produktet av de to første inputene
print("sum",tall1*tall2)
#skriver ut summen av de to siste inputene
print("sum",tall2+tall3)

#28
"""
her er et bedre eksempel for oppgave 26
"""
#tar imot tre input og gjør de om til heltall
tall1 = int(input("tall1:"))
tall2 = int(input("tall2:"))
tall3 = int(input("tall3:"))
#skriver ut produktet av de to første inputene
print("sum",tall1*tall2)
#skriver ut summen av de to siste inputene
print("sum",tall2+tall3)

"""
Løsningsforslag oppgave 29 til 36
Dette er opp til hver enkel person hva som stemmer
og hva man skal gjøre.
"""

#37
"""
påstand = 2>0      uttrykk = 10*10 > 10      variabel = True
navn = False      stortTall = 10**10 < 1     liteTall = 1 < 10**10
"""

#38
alder = 17
print(alder < 3**3)
#mangler --> * i 3**3
#feil --> > betyr at alder er større,
```

```
# men hun vil sjekke om alder er mindre enn 3**3

#39
person1 = 18
person2 = 17
print(person1 != person2)
#mangler --> = i person1 != person2

#40
alder = 26
vairabel = alder > 2**4
print(variabel)

#41
#Per og Pål er to navn. Altså det er ikke påstander, men verdier.

#42
#lagrer tre forskjellige aldre
Alder16 = 16
Alder17 = 17
Alder18 = 18
dinAlder = 26
#sjekker for tre forskjellige aldre
print(Alder16 < dinAlder)
print(Alder17 < dinAlder)
print(Alder18 < dinAlder)









#43
#lagrer aldrene og din alder
Alder16 = 16
Alder17 = 17
```

```
Alder18 = 18
dinAlder = 13
#sjekker hvilke av aldrene du er eldre enn
print(dinAlder>Alder16)
print(dinAlder>Alder17)
print(dinAlder>Alder18)

#44
navn = "Programmering"  #lagrer en tekst til variabelnavnet navn
lengdeNavn = len(navn)  #lagrer lengden av teksten
print(lengdeNavn,navn)  #skriver ut lengden til teksten og teksten

#45
mat = "Matematikk"        #lagrer en tekst til variabelnavnet mat
nat = "Naturfag"          #lagrer en tekst til variabelnavnet nat
print(mat>nat)           #sjekker alfabetisk om mat er før nat
print(len(mat)>len(nat))    #sjekker antall bokstaver om mat er lengre nat

#46
navn = "Hubert Blaine Wolfeschlegelsteinhausenbergerdorff Sr."
print(len(navn))

#47
stedsnavn = "Llanfairpwllgwyngyllgogerychwyrndrobwllllantysiliogogogoch"
print(len(stedsnavn))

#48
navn = "Erik Skaar"
adresse = "Hellerud VGS"
print(len(navn))
print(len(adresse))
```

```
#49
var1 = "Test"
var2 = "test"
var3 = "variabel"
var4 = "operator"
var5 = "påstander"
var1 == var2          ;var2 >= var1      ;var1 > var4
len(var5) > var4      ;var4 != var3      ;len(var5) > len(var3)
len(var2) != len(var1)  ;len(var1) == 4    ;var1 > var5

#50
var1 = "aaaTest"
var2 = "aaatest"
var3 = "aaavariabel"
var4 = "aaaoperator"
var5 = "aaapåstander"
var1 == var2          ;var2 >= var1      ;var1 > var4
len(var5) > var4      ;var4 != var3      ;len(var5) > len(var3)
len(var2) != len(var1)  ;len(var1) == 4    ;var1 > var5


#51
alder = 10
alder2 = 20
if alder+alder2 > 20:            #mangler --> :
   print("Vi er eldre enn 20 år")  #mangler --> ")

#52
#en if-test vil alltid bli kjørt, mens en elif-test
# blir kun kjørt om påstanden i forrige if-test var False

#53
colaPris = 17+2       #+2 pga pant :)
if colaPris*5 < 90:    # Odin bruker > når han vil bruke <
   print("Du kan kjøpe 5 colaer")


#54
```

```
ola = "Ola Nordmann"
espen = "Espen Askeladden"
if len(ola) > len(espen):
   print("ola har lengre navn enn espen")



#55 forklar hva som skjer
#sjekker om Quebec er senere i alfabetet enn Paris
if "Quebec" > "Paris":  #True
   print("Quebec")    #skriver ut Quebec
else:               #dette blir ikke kjørt
   print("Paris")     #fordi påstanden i if-testen
                  #var sann


#56
#lagre prisene i variabler
filmPris = 135
tvPris = 3995
#sjekk om summen av tven og filmene er under 6000
if 6000 > tvPris + 13*filmPris:
   print("Du kan kjøpe TVen og 13 filmer")
else:
   print("Du kan ikke kjøpe TVen og 13 filmer")

#57
#eksempel inntekt på 600'000
inntekt = 600000
#sjekk hvilke skattProsent som skal brukes
if inntekt < 500000:
   skattProsent = 0.3
elif 500000 < inntekt < 1000000:
   skattProsent = 0.4
elif 1000000 < inntekt:
   skattProsent = 0.5

#regn ut skatt og nyInntekt
```

```
skatt = skattProsent*inntekt
nyInntekt = inntekt - skatt
print(inntekt,skatt,nyInntekt)
#refleksjonsspørsmål: Går det an at du får et rart resultat?
#Hint: hva skjer om du skriver inntekt 500'000?









#58
#utvidprogrammet til å bruke input for et fungerende spill
spiller1 = "Stein"
spiller2 = "Papir"

#spill-logikk
if spiller1 == spiller2:
   print("uavgjort")
elif spiller1 == "Stein" and spiller2 == "Saks":
   print("spiller1 vant")
elif spiller1 == "Saks" and spiller2 == "Papir":
   print("spiller1 vant")
elif spiller1 == "Papir" and spiller2 == "Stein":
   print("spiller1 vant")
else:
   print("spiller2 vant")

#59
navn1 = "Stein"
navn2 = "Espen"
navn3 = "Henrik"

#veldig tungvindt.... se alternativ løsning
if navn1 < navn2:
```

```
    if navn3 < navn1:
        print(navn3,navn1,navn2)
    elif navn2 < navn3:
        print(navn1,navn2,navn3)
    else:
        print(navn1,navn3,navn2)
elif navn3 < navn2:
    print(navn3,navn2,navn1)
else:
    if navn3 < navn1:
        print(navn2,navn3,navn1)
    else:
        print(navn2,navn1,navn3)

#dette kan lett gjøres med python sine innbygde funksjoner
l1 = [navn1,navn2,navn3]
l1.sort()
print(l1)



#60
teller = 0
while teller < 10:
    teller = teller + 1 #dette må ha innrykk
    print(teller)      #dette også

#61
teller = 0
while teller < 10: #teller > 10 --> teller < 10
    teller = teller + 2
    print(teller)

#62 hva gjør koden
teller = 1           #startpunkt
while teller < 10:     #påstand --> sluttpunkt er 9
    print("*"*teller)   #skriver ut *'er
    teller = teller + 1 #teller blir større
```

```
#63
#se forrige oppgave
teller = 1
while teller < 10:
   print(" "*(9-teller),"*"*teller)
   teller = teller + 1

#64
#se forrige forrige oppgave
teller = 1
while teller < 10:
   print(" "*teller,
         "*"*(19-(2*teller)),
         " "*teller)
   teller = teller + 1

#65
#start på høyeste tall
teller = 5
while teller >= 0:
   print(teller)
   teller -= 1    #-1 for hver gang




#65 - alternativ løsning
#start på 0
teller = 0
while teller <= 5:
   print(5-teller) #skriv ut 5-teller -- teller er 0,1,2,3,4 aldri 5
   teller += 1    #+1 for hver gang

#66
alder = 26        #startpunkt er 26 for meg
while alder < 25:   #sjekk om alder er mindre enn 25
```

```
   alder +=1      #pluss på 1 til alder om påstanden er sann


#67
#se kommentarer fra forrige oppgave
#legger til en variabel som skal telle hvor mange år det tar
# starter på 0
teller = 0
alder = 26
while alder < 25:
   alder +=1
   teller +=1
   #legg til et år for hver gang while-løkken sin kode kjører
print(teller)


#68
summen = 0
teller = 0     #starter på 0
while teller < 10:
   teller += 1       #legg til 1 før du legger til teller på sum
   summen += teller    #legg til teller på sum
print(summen)

#69
oddetall = 0    #sum oddetall = 0
partall = 0     #sum partall  = 0
teller = 1     #start tall er 1
while teller < 100:
   oddetall += teller  #legg til 1 første gang   -oddetall
   partall += teller+1 #legg til 1+1 første gang -partall
   teller += 2       #stig med 2
print(oddetall,partall)
#det er mange muligheter her. Dette er kun et forslag.

#70
#range starter automatisk på 0 og
#      stiger automatisk med 1 for hver gang
#      slutter på tallet før 5, altså 4
```

```
for i in range(5):
    print(i)

#71
#range starter på 0 og
#       stiger automatisk med 1 for hver gang
#       slutter på tallet før 10, altså 9
for i in range(0,10):
    print(i)

#72
#range starter på 5 og
#       stiger med 2 for hver gang
#       slutter på tallet før 10, altså 9
for i in range(5,10,2):
    print(i)

#73
teller = 5 #startverdi settes til 5
while teller < 10: #slutt når påstanden er false
                #altså at teller er større enn 10
    print(teller)   #skriv ut teller
    teller += 2     #legg til 2 på teller

#74
#lager en liste
liste1 = [2,3,5,7]
#går igjennom listen, et element av gangen
for element in liste1:
    #skriver ut elementet
    print(element)
```

```
#75
#index    0 1 2 3
liste1 = [2,3,5,7]
#vi bruker range 4, fordi index går fra 0 til og med 3
for i in range(4): #alternativt - range(len(liste1))
    #skriv ut elementet på indeksen sin plass
    print(liste1[i])

#alternativt
#hvis du ønsker å lage løkken men uten en liste
a,c = -1/6,1/6
for i in range(4):
    #legg merke til at dette er stygg kode
    #formelen er hentet fra regressjon på 2,3,5,7
    print(int(a*i**3 + i**2 + c*i + 2))

#while-løkke
liste1 = [2,3,5,7]
index = 0
#vi skal gå igjennom indeksene til listen
while index < 4: #alternativt index < len(liste1)
    print(liste1[index])
    index += 1 #manuelt legg til 1 på indeksen

#76
#lager en liste med 5 elementer
liste1 = [0,1,2,3,4]
#lager en tekst 01234, bestående av 5 tegn
tekst = "01234"
print(len(liste1))  #skriver ut antall elementer --> 5
print(len(tekst))   #skriver ut antall tegn --> 5

#77
dna = "ATCG"
#bruker index som variabelnavn i for-løkken
```

```python
for index in range(len(dna)):
    print(dna[index])   #mangler --> index i [...]

#78
dna = "ATCG"
#bruker element som variabelnavn for hver ting i dna-"listen"
for element in dna:
    print(element)  #skriver ut elementet


#79   #a     #lag en variabel
bokstav = "T"
if bokstav == "A":  #bruk == for å sjekke om den er lik "A"
    print("A")     #Hvis de er like, så skriver vi ut "A"

#b     #lagrer en tekst
tekst = "Jeg heter Erik"
#bruker bokstav som variabelnavn for hvert tegn i tekst
for bokstav in tekst:
    #går igjennom teksten et tegn av gangen
    print(bokstav)     #skriver ut hvert tegn i tekst

#c     #lagrer en tekst
tekst = "Jeg heter Erik"
#bruker bokstav som variabelnavn for hvert tegn i tekst
for bokstav in tekst:
    #sjekker om bokstaven er lik "A"
    if bokstav == "A":
        #hvis ja, så skriver vi ut A
        print("A")

#80 naturfag utfordring
#lager en tekst i variabelnavnet tekst
tekst = "ATCG blir TAGC"
for bokstav in tekst:
    if bokstav == "A":        #sjekk om bokstav er lik "A"
        print("T")
    #gjør det for de fire bokstavene
```

```
    elif bokstav == "T":
        print("A")
    #vi kan bruke if istednfor elif om vi vil
    elif bokstav == "C":
        print("G")
    elif bokstav == "G":
        print("C")

#81
#tar imot en brukerinput og gjør det til et tall
# dette kan kræsje om man skriver noe som ikke er et tall
tall = int(input("Skriv et tall:"))
#sjekk om tallet er fra og med 0 til og med 10
if 0 <= tall <= 10:
    #lag en for-løkke som går igjennom 1,2,...,9,10
    for i in range(1,11,1):
        print(i,tall*i) #skriv ut telleren og dens gangestykke
```