# Introduksjon til programmering

Andøya Space

.no esero

## Kort om aktiviteten

Denne aktiviteten skal introdusere og gi eleven forståelse for programmering, algoritmisk tenkning og tre konsepter som går igjen på tvers av programmeringsspråk; variabler, løkker og vilkår.

Andøya Mission Control - Oppdrag: Mars er et læringsverktøy som tar i bruk denne måten å tenke på, og lar eleven lære om programmering og algoritmisk tenkning gjennom praktiske og konkrete oppgaver, med verdensrommet som tema.

## Mål fra Læreplanen

Matematikk:

- Etter 6.trinn: bruke variablar, lykkjer, vilkår og funksjonar i programmering til å utforske geometriske figurar og mønster
- Etter 7. trinn: bruke programmering til å utforske data i tabellar og datasett
- Etter 8. trinn: utforske korleis algoritmar kan skapast, testast og forbetrast ved hjelp av programmering
- Etter 10. trinn: utforske matematiske eigenskapar og samanhengar ved å bruke programmering

Valgfag programmering (PRG1-01)

- bruke flere programmeringsspråk der minst ett er tekstbasert
- bruke grunnleggende prinsipper i programmering, slik som løkker, tester, variabler, funksjoner og enkel brukerinteraksjon
- utvikle og feilsøke programmer som løser definerte problemer, inkludert realfaglige problemstillinger og kontrollering eller simulering av fysiske objekter
- omgjøre problemer til konkrete delproblemer, vurdere hvilke delproblemer som lar seg løse digitalt, og utforme løsninger for disse
- dokumentere og forklare programkode gjennom å skrive hensiktsmessige kommentarer og ved å presentere egen og andres kode

# Innhold

Kort om aktiviteten ..... 1
Mål fra Læreplanen ..... 1
Lærerveiledning ..... 3
Om Algoritmisk tekning og programmering ..... 3
Datamaskiners og menneskers oppgaver ..... 3
Instrukser til roveren ..... 3
Variabler, løkker og vilkår ..... 4
Algoritmer og funksjoner ..... 5
Skrive kode for hånd eller byggeklosser ..... 5
Programmering i Oppdrag: Mars ..... 7
Reelle programmeringsverktøy ..... 10
Etterarbeid med reelt programmeringsspråk ..... 11
Kodeøvelse 1: Variabler ..... 12
Kodeøvelse 2: Vilkår ..... 13
Kodeøvelse 3: Løkker ..... 14
Kodeøvelse 4: Funksjoner ..... 16
Kodeøvelse 5: Alt kombinert ..... 17
Kilder ..... 20

# Lærerveiledning

## Om Algoritmisk tenkning og programmering

Ifølge Udir er **algoritmisk tenkning** en problemløsningsmetode. Litt forenklet kan vi si at det er «å tenke som en informatiker» når vi skal løse problemer eller oppgaver. En informatiker er en som må vurdere hvilke steg som skal til for å løse et problem, og bruke sin teknologiske kompetanse for å få en datamaskin til å løse deler av eller hele problemet. Det betyr at en informatiker bør ha en forståelse av hva slags oppgaver teknologi kan hjelpe til med å løse og hva som bør overlates til mennesker å gjøre.

**Programmering** er det informatikeren gjør når hun/han setter sammen eller skriver instrukser og mater datamaskinen med dem. Instrukser eller programkode kan sees på som en oppskrift for hvordan datamaskinen kan løse deler eller hele utfordringen vi har bestemt at den skal hjelpe oss med.

## Datamaskiners og menneskers oppgaver

Andøya Mission Control – Oppdrag: Mars er et læringsverktøy som tar disse to konseptene og gjør de praktisk og relevant.

I Oppdrag: Mars må gruppene sette seg inn i hva Mars-roveren skal hjelpe dem med, som de selv ikke kan gjøre. Oppgaven til elevene er å forberede en rover med instrukser, sende den til Mars og få den til å returnere nyttig informasjon som den samler inn på Mars-overflaten.

Dette er en oppgave vi mennesker ikke kan gjøre selv per i dag og da trenger vi hjelp fra datamaskiner. En Rover er en robot med en datamaskin som hjerne.

Vi mennesker kan bygge en rover, gi den nøyaktige instrukser, også kan den reise til en fjern planet og utføre oppgavene for oss. Her kommer tydelig skillet på hva vi kan gjøre og hva en datamaskin kan bidra med.

## Instrukser til roveren

Algoritmisk tenkning innebærer å bryte ned komplekse problem til mindre, mer håndterlige delproblemer som lar seg løse. Det inkluderer å organisere og analysere informasjon på en logisk måte og å lage fremgangsmåter (algoritmer) for å komme fram til ønsket løsning.

Roveren som skal programmeres med bestemte instrukser har et hovedoppdrag; Å samle inn data fra Mars og sende det tilbake til jorden. Denne utfordringen kan og må deles opp i mindre oppgaver som hver av elevgruppene tar for seg.

I Oppdrag: Mars jobber elevene direkte fra klasserommet og i små grupper. Gruppene må konsentrere seg om deloppgaver som roveren skal utføre etter landing og utkjøring på planetens overflate.

Roveren har ulike sensorer, instrumenter og teknologi som hver krever ulike fremgangsmåter eller algoritmer. Gruppene får ansvar for ulike algoritmer de må sette sammen. Når alle algoritmene er bygd opp kan roveren samlet utføre de i en planlagt rekkefølge eller sekvens.

## Variabler, løkker og vilkår

I alle programmeringsspråk og instruksjonssett er det noen ting til felles, men som vil se litt forskjellige ut fra språk til språk. Litt som ulike menneskelige dialekter og språk.

Det er tre konsepter som Udir ønsker at elevene skal lære; variabler, løkker og vilkår. Dette er grunnleggende for alle språk.

Kort fortalt er variabler en verdi tilordnet et ord eller en bokstav. For eksempel kan vi si at «variabel1» skal inneholde teksten «jeg er en variabel» eller tallet «312345». I stedet for å bruke selve teksten eller tallet som du kanskje vil bruke mange ganger, kan du bruke en variabel som da vil peke på det innholdet du har gitt den, altså teksten eller tallet. Innholdet til variabelen kan også endres og du kan ha mange variabler av ulike typer, om de er tall, bokstaver, en dato eller noe annet. Slike typer er også litt typisk for programmeringsspråk å forhåndsdefinere. Altså, du har en bestemt rekke ulike typer du kan forholde deg til å bruke i algoritmene.

Løkker er en gjentagende hendelse. Ønsker du å gi roveren beskjed om å gå fremover eller gjøre en måling eller andre mekaniske aktiviteter, kan du velge å gjøre det én gang eller flere ganger. Skal du gjøre samme aktiviteten eller instruksen flere ganger kan du bruke løkker for å gjenta noe et gitt antall ganger, eller inntil noe spesielt inntreffer. For eksempel kan en person som spør om veien til nærmeste matbutikk få et svar eller en instruks om å gå fremover til han/hun kommer til et kryss (og få videre instrukser deretter). Uten en løkke ville instruksene kanskje vært å gå frem, gå frem, gå frem, gå frem, gå frem mange

ganger kanskje, noe som ville blitt vanskelig å huske for personen som ville til butikken.

Vilkår er en logisk tenkemåte om å se etter en bestemt situasjon og hvis den inntreffer så skal noe skje.

For eksempel kunne personen som ville til butikken fått en instruks om å gå frem og sjekke om du står i et kryss. Hvis ja, roter 90 grader og gå frem. Hvis ikke, fortsett å gå frem. Slike vilkår eller tester kan brukes i en løkke også, noe som vil føles naturlig. Fortsett å gå frem til du står i et kryss. Vilkår kan også baseres på andre ting, for eksempel om roveren skal ta et bilde hvert minutt, eller om sende tilbake data bare hvis satellittforbindelsen er i orden og hvis ikke, forsøke å etablere den.

## Algoritmer og funksjoner

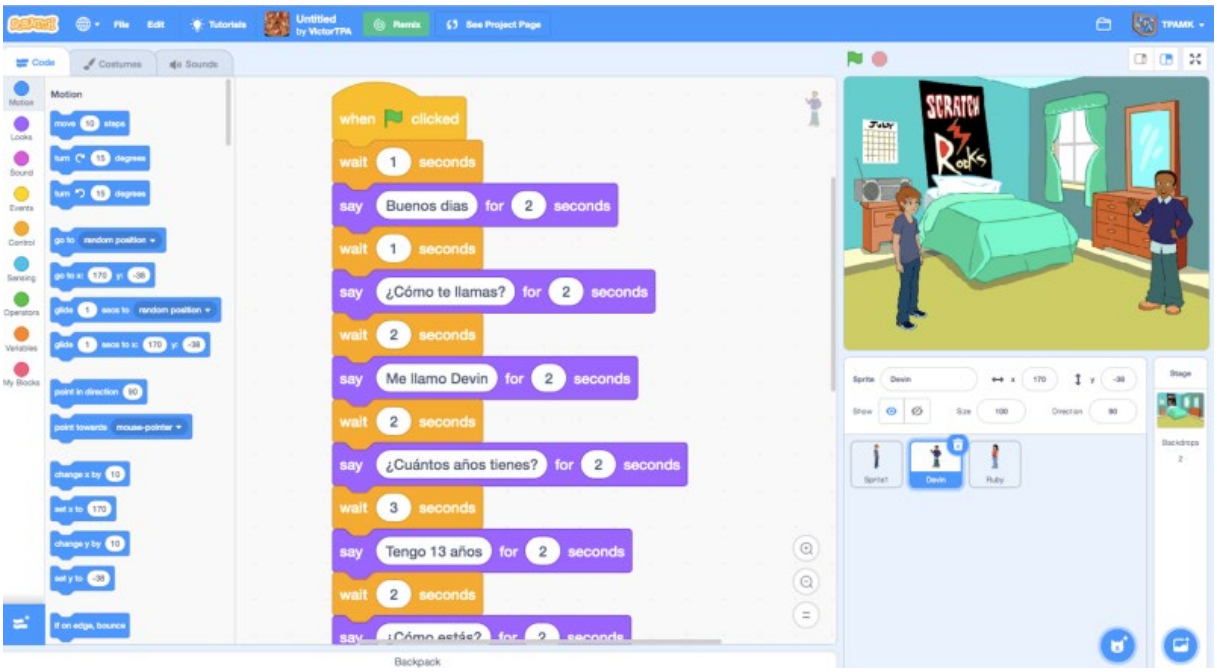
Algoritmene som elevene må sette sammen vil bruke løkker, vilkår og variabler. I et programmeringsspråk som Python eller Java vil man skrive instruksjoner i en ordnet rekkefølge og i enkelte samlede grupper. Noen instrukser kan samles i en gruppe som igjen utfører en større oppgave. En slik samling kan utføre samme oppgaven flere ganger, men for eksempel med litt ulike parametere eller bestemmelser.

I flere språk kalles en slik gruppe funksjoner eller prosedyrer. For eksempel kunne roveren blitt utstyrt med en funksjon som kan gå frem, snu og ta et bilde. Da kan vi si at funksjonen har tre instrukser i seg, gå frem, snu og ta bilde. Men vi sier ingenting om hvor langt frem den skal gå eller hvor mye den skal snu seg.

Vi kan bestemme at denne funksjonen skal kjøre eller utføres på bestemte tidspunkt, steder eller intervaller. På de ulike gangene de kjøres, skal de bruke ulike verdier i instruksene sine. Når vi kjører funksjonen gir vi samtidig informasjon om hvor lenge den skal kjøre frem og hvor mange grader den skal snu.

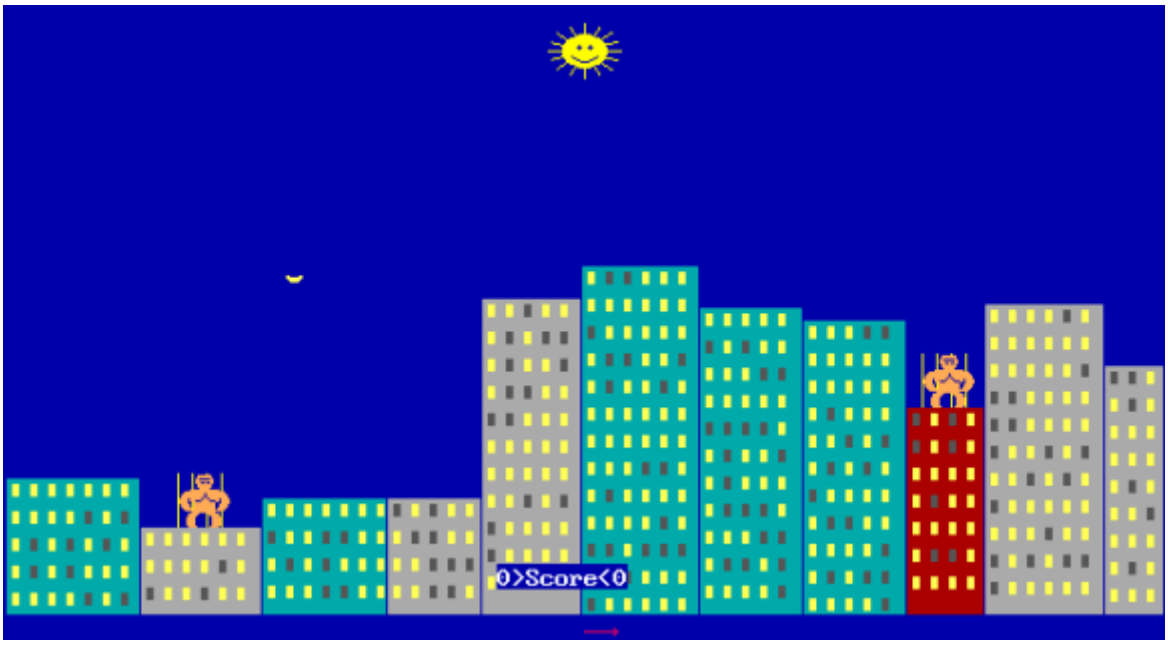
## Skrive kode for hånd eller byggeklosser

Mange vil helst introdusere programmering for barn og unge gjennom lego-lignende programmeringsspråk som ser ut som puslespill eller byggeklosser. Dette er en fin måte å forstå sammenhenger og konsepter, men ikke så veldig virkelighetsnært.




På 80- og 90-tallet satt unger foran en TV med datamaskiner innebygd i et tastatur, enten det var den britiske forgjengeren til Micro:bit, Amiga eller Commodore 64.

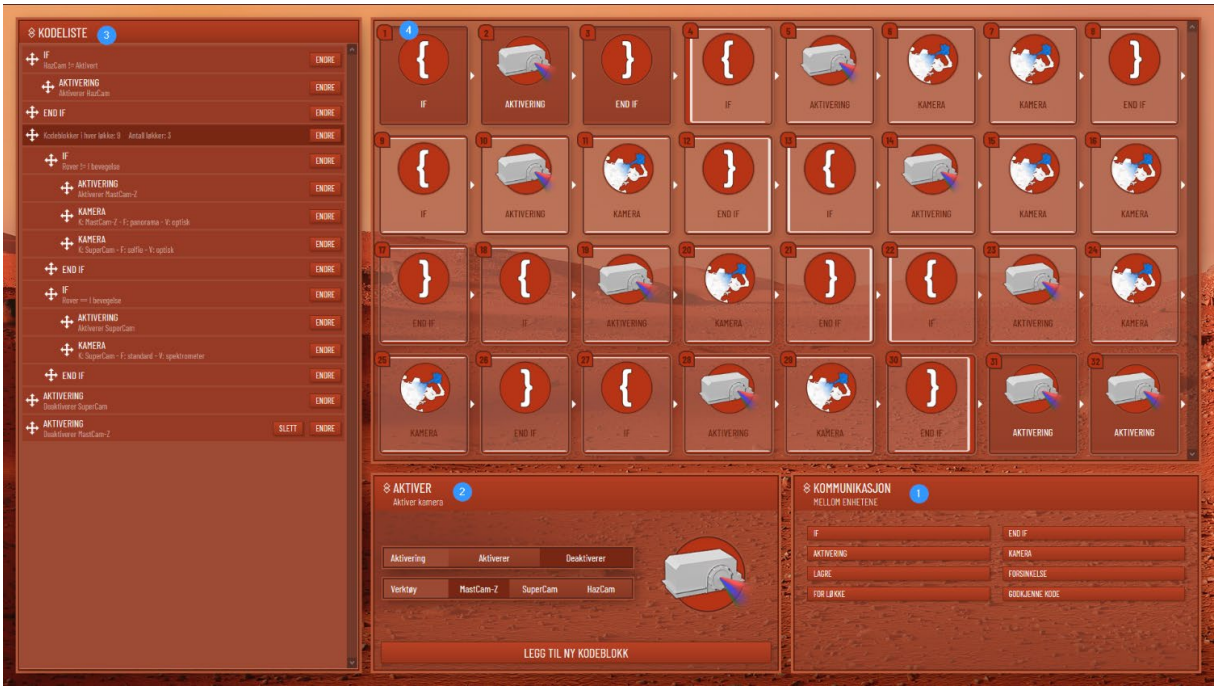
Da utforsket de programmering i et tekstbasert grensesnitt. De kjøpte datamagasiner i kiosken eller på butikken som hadde algoritmer og spill i kode skrevet ut over flere sider. Ungdommene kunne bruke timer på å skrive inn koden som var i bladet, for deretter kjøre den på sin maskin, og gjennom dette lærte de seg programmering.




Det er kanskje ikke én rett måte å lære seg å programmere på.

## Programmering i Oppdrag: Mars

I Oppdrag: Mars bruker vi et blokkbasert, grafisk grensesnitt for å bygge opp programkoden som roveren skal klargjøres med, hvor variabler, løkker og vilkår visualiseres.

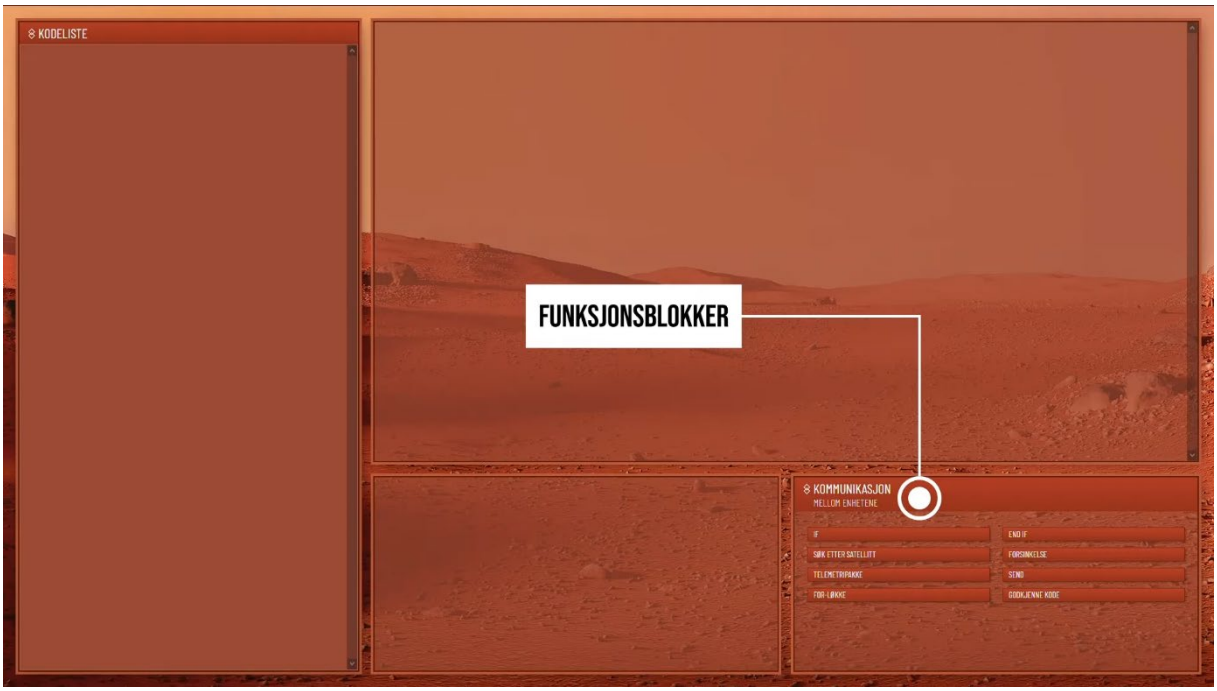



Derfor er det viktig å bli litt kjent med dette før elevene skal begi seg ut på et oppdrag med Andøya Mission Control.

Det er fire områder/vinduer i kontrollpanelet når det programmeres:

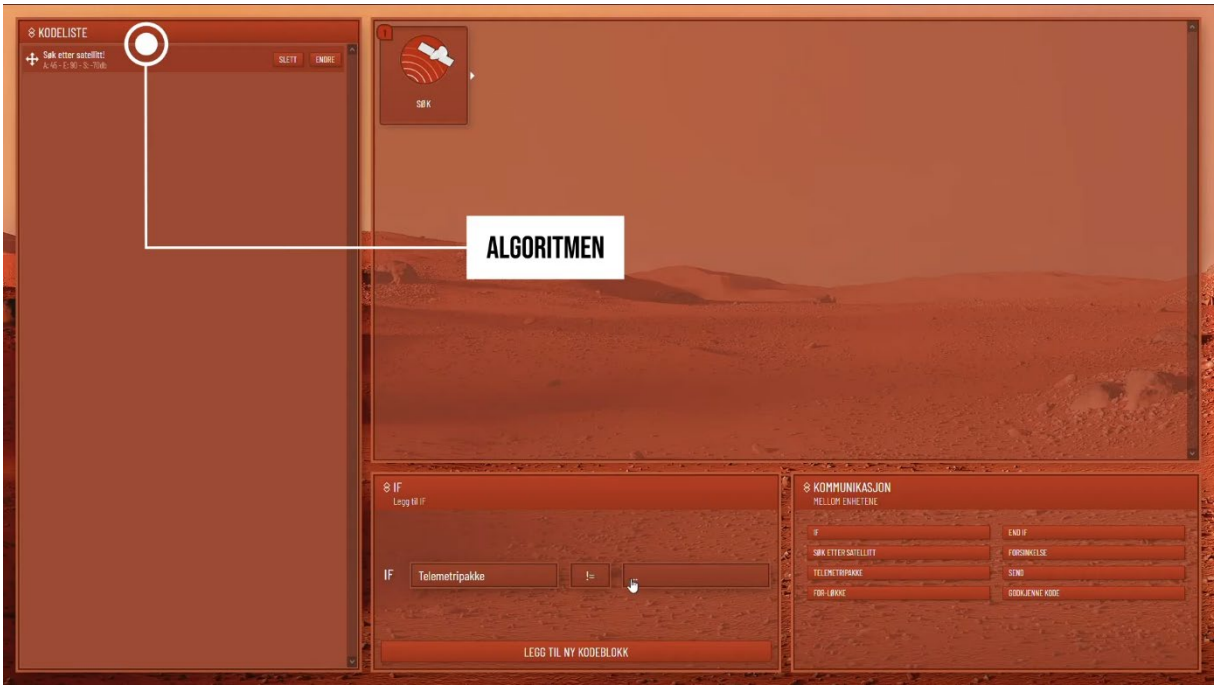
1. Kodeblokker tilgjengelig for denne gruppen
2. Parameter for en valgt kodeblokk
3. Kodelisten etter hvert som blokker blir lagt til listen
4. Ferdig kompilert algoritme som skal sendes til roveren

Du starter ved å velge en blokk i blokk-/funksjonslisten.




Deretter velger man de parametrene man ønsker og trykker på legg til kodeblokk.

Blokken eller funksjonen og dens parametere blir så synlig i kodelisten. Denne kodelisten bygges opp etter hva som er behovet og ønsket for gruppen.




En beskrivelse av hva som må tas hensyn til er gitt den enkelte gruppe i instruksjonsmanualene. Det er viktig å gjøre seg kjent med de i forkant av oppdraget.

Etter hvert som kodelisten bygges opp, vil du se i den ferdige kompilerte algoritmen som sendes til roveren.

Kodelisten er mer leselig for mennesker og kalles gjerne pseudokode. Her brukes innrykk for å øke leseligheten i koden og har løkker og vilkår blitt brukt, vil innrykkene være synlige.

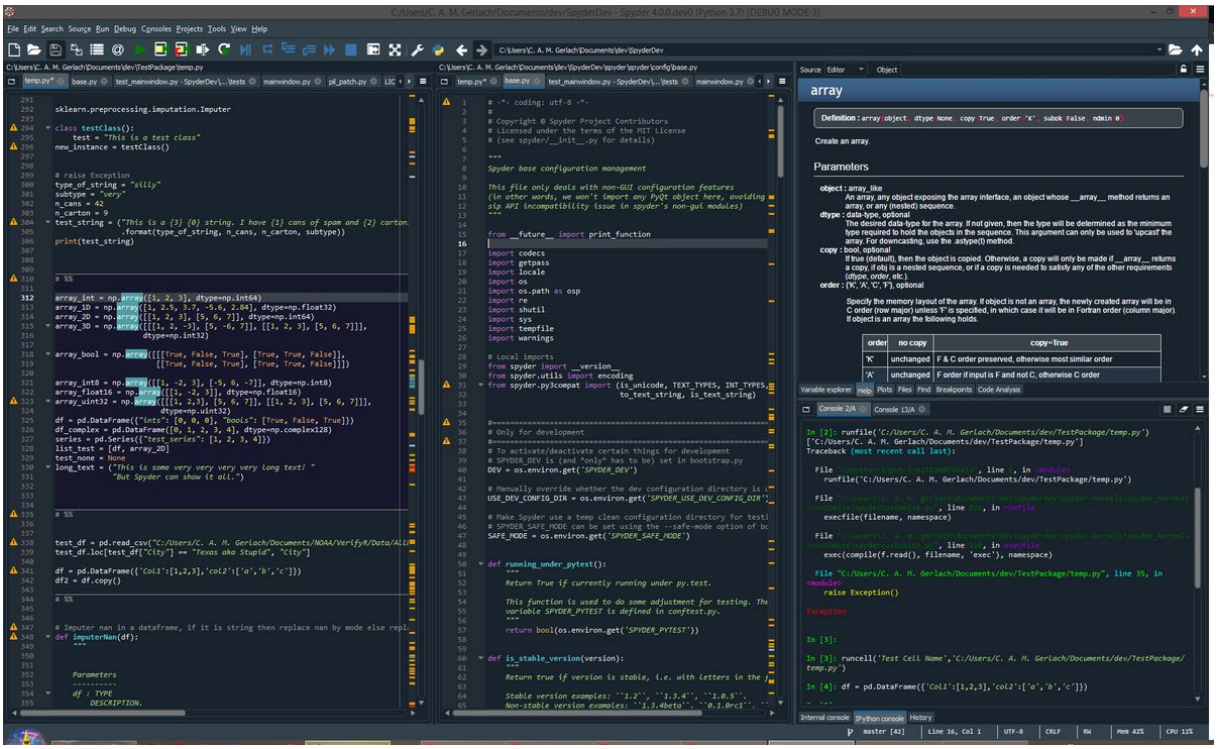
På YouTube kan du se en video av hvordan dette verktøyet ser ut i bruk:

https://youtu.be/c-_4VkeqgCw

## Reelle programmeringsverktøy

Folk som jobber som utviklere og programmerere til daglig, eller koder mye, har gjerne installert egne programmer som de skriver programkode i. Dette kalles gjerne en editor eller en IDE (integrated development environment) og er på en måte en slags tekstbehandler, men for koding.

Slike verktøy har innebygd funksjonalitet for å feilsøke i koden og gjør det lettere for de som jobber til daglig med slikt.



**NB!** Det er ikke nødvendig for denne aktiviteten å installere noe som helst, for det finnes flere enkle nettverktøy som lar deg skrive Python-kode og kjøre koden direkte i en nettleser. Her er et par utvalgte verktøy som fungerer bra:

- https://www.python.org/shell/
  - Fin til å skrive en og en linje om gangen
- https://repl.it/languages/Python3
  - Denne kan brukes til å lime inn kode fra for eksempel dette dokumentet og kjøre det direkte.

Til øvelsene i etterarbeidet kan det derimot være nødvendig å ha installert programvare for å arbeide på datafiler og produsere grafiske fremstillinger, grafer og lignende.

## Etterarbeid med reelt programmeringsspråk

Python er et av de mest populære programmeringsspråkene i dag og med god grunn. Det er mye brukt, lettere å lese enn mange andre språk og kan løse det aller meste av oppgaver.

For å bygge videre på forståelsen og arbeidet som gjøres i Oppdrag: Mars legges det opp til at elevene skal få med seg innsamlet data fra roveren etter fullført oppdrag. Disse dataene kan så fremstilles ved hjelp av Python eller valgfritt programmeringsspråk / verktøy, for eksempel et regneark (Excel eller lignende).

Når du har fått fremstilt data på en grafisk måte er det lettere å analysere funnene dine fra oppdraget.

En annen oppgave i etterkant er å skrive koden du jobbet med i oppdraget om til Python-sjargong.

- Etterarbeid 1: Analyse av data ved hjelp av programmering
- Etterarbeid 2: Rapportskriving
- Etterarbeid 3: Oversette pseudokoden/kodelisten fra oppdraget til Python-kode (definere tomme funksjoner, men reelle løkker og vilkår)

Øvelsene i denne aktiviteten bruker altså Python som språk og går gjennom de ulike konseptene som Udir legger opp til i læreplanen. De samme konseptene blir deretter brukt i oppdragets kodeverktøy.

## Kodeøvelse 1: Variabler

En variabel er en lagringsplass for dataverdier, for eksempel tekst eller tall. En slik lagringsplass kan når som helst tas i bruk og vi kan hente ut verdien av en eller flere slike plasser samtidig.

I Python oppretter vi en variabel på følgende måte:

```
Variabelnavn = dataverdi
```

Når vi ønsker å lagre tall, kan vi gjøre det direkte, men skal vi lagre tekst eller setninger, må vi binde det sammen med hermetegn.

I matematikken brukes = (likhetstegnet) for å si at noe er lik noe. I programmeringsspråk brukes likhetstegnet (=) litt annerledes. Når vi under bruker = sier vi at vi tilordner det foran likhetstegnet med en verdi bak likhetstegnet.

Vi kan lage to variabler, en for tekst og en for tall. Deretter kan vi få datamaskinen til å skrive disse verdiene ut på skjermen:

```
drivstoff_tekst = "Gjenstående drivstoff:"
drivstoff_mengde = 24

print(drivstoff_tekst, drivstoff_mengde)
```

Her tar vi også i bruk en mye brukt funksjon i Python som heter print(). Funksjoner kalles eller brukes ved å angi navnet på funksjonen og legge til parenteser

```
print("tekst eller variabler som skal skrives ut på skjermen")
```

Hvis du vil at det skal komme en ny tom linje etter at print() har skrevet ut noe på skjermen, kan du legge til \n på slutten av teksten, slik:

```
print("tekst eller variabler som skal skrives ut på skjermen\n")
```

Bruk en av de nettbaserte Python-editorene og prøv å lage variablene over og få de skrevet ut etterpå.

*HUSK AT variabelnavn ikke kan ha mellomrom i seg, men bruk _ eller – som skilletegn dersom du vil ha det.*

Variabler kan også endres i ettertid, ved å tilordne en ny dataverdi til samme variabelnavn, på samme måte som du opprettet den. Da overskriver du lagringsplassen med nytt innhold.

Dette kan for eksempel være roveren sin drivstoffmengde som blir mindre over tid. Det skal vi se på i neste oppgave.

## Kodeøvelse 2: Vilkår

Vilkår er en sjekk om en situasjon er på en eller annen måte. I programmering brukes if-tester for å sjekke om vilkårene er oppfylt. I dagligtale kan du spørre: «Hvis du skal på butikken, husk å handle melk». Det er ikke så ulikt i kode:

```
if skalpåbutikken == true:
  handlemelk
```

Der bruker vi en if-test for å sjekke om vilkåret «skal på butikken» er sant eller ikke.

Legg merke til strukturen i eksemplet over. Vi bruker et kolon-tegn bak sjekken for å indikere at det som følger skal skje dersom vilkåret inntreffer. Legg merke til at kodelinjen under if-linja er forskjøvet mot høyre. Dette kalles et innrykk, og forteller koden at denne linja hører til if-testen. Alle linjer som skal høre til if-

testen må være innrykket, gjerne med to eller fire mellomrom eller en tabulator. Slike innrykk er viktige i Python for å angi hvilke kodelinjer som hører sammen.

*HUSK PÅ å alltid bruke samme type innrykk, enten det er to eller fire mellomrom eller tabulator, ellers vil Python stoppe og rapportere at noe er feil i koden.*

Sannhetssjekken eller selve testen bruker == for å sammenligne om «skalpåbutikken» er lik «true». I forrige øvelse så vi at når vi under bruker = sier vi at vi tilordner det foran likhetstegnet med en verdi bak likhetstegnet. Når vi skal sammenligne en variabel med en annen eller sjekke om noen vilkår er til stede, bruker vi i programmeringsspråk ulike sammenligningsoperatører.

Det er noen få slike logiske betingelsen vi kan bruke, både i tester, men også i løkker:

| | |
|---|---|
| == | For å sjekke om noe er identisk med noe annet, **det samme som** |
| != | For å sjekke om noe ikke er identisk, **ikke det samme som** |
| > | For å sjekke om noe er **større enn** noe annet |
| < | For å sjekke om noe er **mindre enn** noe annet |
| <= | For å sjekke om noe er **mindre enn eller lik** |
| => | For å sjekke om noe er **større enn eller lik** |

Ved å bruke mindre enn-tegnet < kan vi sjekke om en verdi har gått under et bestemt nivå, for eksempel om roveren har mistet for mye drivstoff og må avbryte:

```
if drivstoff_mengde < 3:
  print("Roveren er nesten tom for drivstoff, avbryt!")
```

Bruk en nettbasert Python-editor og prøv å utvide koden din med en slik kode, plassert under koden fra den første øvelsen.

## Kodeøvelse 3: Løkker

Til nå har vi opprettet en variabel for å huske hvor mye drivstoff vi har. Vi fikk roveren til å skrive ut mengden på skjermen. Vi la også inn en if-test for å sjekke om nivået var for lavt.

For å gjøre dette litt mer virkelig, må vi forsøke å simulere at roveren faktisk kjører bortover og bruker drivstoff. Dette kan vi gjøre med en for-løkke.

Python bruker to typer løkker, *for* og *while*.

- En for-løkke brukes for å gå gjennom en bestemt rekke eller mengde med data. Se for deg at noen sier: «Rydd fem ting på rommet ditt, så kan du gå ut» eller «etter at du har tømt søpla, vasket gulvet og ryddet rommet ditt, kan du går ut».
- En while-løkke brukes for å holde på med noe så lenge eller inntil noen vilkår inntreffer. Dette er som å si til noen: «så lenge du du ikke har ryddet rommet ditt, får du ikke gå ut» eller «før du kan gå ut, må du rydde rommet ditt».

Med en for-løkke kan vi gradvis minke drivstoffet og med en if-test hele tiden passe på at det ikke er for lite. Hvis det blir for lite må vi avbryte kjøringen:

```
drivstoff_mengde = 24

for drivstoff in range(drivstoff_mengde, 0, -1):
  print("Gjenstående drivstoff:", drivstoff)

  if drivstoff < 3:
    print("Roveren er nesten tom for drivstoff, avbryter!")
    break
```

Selve kommandoen for å sette i gang for-løkken settes opp for å gå gjennom en gitt mengde data, i dette tilfellet drivstoffmengden.

```
for drivstoff in range(drivstoff_mengde, 0, -1):
```

*range()* er en innebygd Python-funksjon som vi kan bruke til dette. Vi simulerer at roveren bruker drivstoff ved å gi range() hvor mye drivstoff som er der, hva som er tomt, altså null og at den teller nedover -1 fra full tank og ned til null:

```
range(drivstoff_mengde, 0, -1):
```

I løkken har vi også tatt med if-testen, for å sjekke underveis i simuleringen. Vi introduserer dessuten en Python-metode for å avbryte løkken; *break*, dersom det vilkåret skulle inntreffe.

*HUSK AT oppbyggingen til en løkke må på samme måte som en if-test ha et kolon-tegn og et innrykk på det som skal skje i selve løkken.*

---

Bruk din favoritt Python-editor og pakk inn if-testen din med en slik for-løkke, og sett opp løkken.

---

## Kodeøvelse 4: Funksjoner

En funksjon er samling av kode som bare kjøres når funksjonen kalles. Vi har sett på en innebygd funksjon *print()* som vi tok i bruk på et bestemt tidspunkt. Denne funksjonen er definert et annet sted og inneholder kode som vi ikke ser i vår egen kode. Men vi vet at det tar tekst eller variabler som parameter (det vi legger inn i parentesen) og skriver det ut på skjermen.

Nå skal vi lage en egen funksjon som gjør det vi har gjort i kodeøvelse 2 og 3.

Vi oppretter og definerer en funksjon i Python på følgende måte:

```
def funksjonsnavn(parametre):
```

Legg merke til kolon-tegnet som er bak definisjonen (def er kort for define på engelsk) og det som skal være i funksjonen må ha innrykk, på samme måte som løkker og if-tester.

Vi gir funksjonen et fornuftig navn og bestemmer at den skal kunne bli kalt opp med en parameter. Denne parameteren kan være en verdi eller en variabel vi kan gi til funksjonen når vi kaller den. Vi kommer til å gi den *drivstoffmengden* som parameter, som funksjonen deretter kan bruke i koden sin.

Vi må velge et internt parameternavn som kun blir brukt inne i funksjonen som ikke er det samme som variabelen vi sender inn i funksjonen. Det kan ellers være stort sett hva som helst, men for å gjøre det kort og enkelt, kaller jeg den x.

```
def rover_drivstoff_funksjon(x):
  print("Roveren begynner å kjøre\n")
  for drivstoff in range(x, 0, -1):

    if drivstoff < 3:
      print("Roveren er nesten tom for drivstoff, avbryter!")
      break

    print("Gjenstående drivstoff:", drivstoff)
```

Da må vi også endre for-løkken fra tidligere, slik at den også bruker x inne i funksjonen:

```
for drivstoff in range(x, 0, -1):
```

Legg merke til at det er tre nivåer nå. En if-test i en for-løkke i en funksjon. Alt er innrykket. Det fine med Python er at slike innrykk viser samtidig strukturen i koden og det blir lettere å lese hva vi har programmert.

Nå har vi en funksjon som tar en forventet og påkrevd parameter, nemlig drivstoffmengden til roveren.

Når vi nå ønsker å bruke funksjonen kaller vi den opp slik:

```
rover_drivstoff_funksjon(drivstoff_mengde)
```

---

Pakk inn for-løkken inkludert if-testen din i en funksjon. Husk å ta med en *break*. Utenfor, under funksjonen (ikke innrykket) kan du kalle opp funksjonen med drivstoffmengdevariabelen som parameter.

---

## Kodeøvelse 5: Alt kombinert

Nå har vi sett på variabler, vilkår, løkker og funksjoner. Du har kanskje allerede satt sammen alt til en algoritme og kjørt den i en Python-editor.

Det er et par ting til vi kan gjøre for å gjøre algoritmen mer troverdig og lettere å lese. Vi kan legge inn noen kommentarer i koden for å fortelle oss som leser den hva de ulike tingene gjør. Dette vil bli ignorert når koden kjøres, men gjør det lettere å lese. Firkanttegnet # brukes for å legge inn kommentarer, enten på egen linje eller bak en instruks eller linje:

```
# dette er en kommentar
```

```
print("kommentarer er veldig nyttig") # dette er også en kommentar
```

Det andre vi skal introdusere til slutt er en forsinkelsesteknikk for å se at drivstoffet faktisk minker gradvis og ikke alt på en gang som det har til nå.

Vi må hente inn ekstra funksjonalitet og gjør det ved å legge inn i toppen

```
import time
```

*time* er et ekstra sett med funksjoner, deriblant funksjonen *sleep()* som kan brukes for å pause kjøringen av koden med gitt antall sekunder.

```
# Vi må importere ekstra funksjonalitet
import time

# Vi lager variabel for drivstofftanken
drivstoff_mengde = 24

def rover_drivstoff_funksjon(x):

  # Skriv ut at vi begynner å kjøre
  print("Roveren begynner å kjøre\n")

  # Vent ett sekund før vi begynner
  time.sleep(1)

  # En løkke som simulerer drivstoffbruken til roveren
  for drivstoff in range(x, 0, -1):

    # Vi sjekker om drivstoffnivået er for lavt
    if drivstoff < 3:
      print("Roveren er nesten tom for drivstoff, avbryter!")
      # gå ut av løkken
      break

    print("Gjenstående drivstoff:", drivstoff)

    # Vent ett sekund før drivstoffet minker
    time.sleep(1)

# Vi kjører funksjonen vår
rover_drivstoff_funksjon(drivstoff_mengde)
```

Nå ser du oppbyggingen til et typisk Python-program eller script som det også kan kalles. Dette er en simulering som vi kunne lagret som en fil og gitt den for eksempel navnet rover-simulering.py (.py for å si at dette er Python kildekode).

Deretter kunne dette programmet kjøres i en datamaskin for å gjøre simuleringen.

Prøv den kombinerte koden, med ulike mengder drivstoff, hvor mye som er igjen før den avbryter og raskere eller tregere forsinkelser.

Gratulerer, du har skrevet ditt første dataprogram, en roversimulering.

## Kilder


- https://www.esero.no/ressurser/grunnskolen/#amc
- https://www.udir.no/kvalitet-og-kompetanse/profesjonsfaglig-digital-kompetanse/algoritmisk-tenkning/
- https://www.udir.no/laring-og-trivsel/lareplanverket/veiledning-lp/valgfag-programmering/ordliste/
- https://www.udir.no/kl06/PRG1-01/Hele/Kompetansemaal/programmering
- https://www.youtube.com/watch?v=c-_4VkeqgCw