AKTIVITET

Å kode sammen – livekoding

Klasseromaktivitet for skoleelever

.no esero

Andøya Space

Kort om aktiviteten

Programmering og algoritmisk tenkning er synliggjort i fagene på ulikt vis i Læreplanverket for Kunnskapsløftet 2020. Elevene får en introduksjon til sentrale konsepter og grunnstrukturer i programmering i matematikkfaget, men de skal lære og anvende kompetansen også i andre fag. Det er eksplisitte kompetansemål i naturfag, kunst og håndverk og musikk. Programmering kan også styrke andre fag, som for eksempel samfunnsfag og norsk.

Som en introduksjon til programmering kan det være lurt å se på konseptene på en utforskende og litt enklere måte for å få med alle elevene. La elevene ha det gøy mens de utforsker dette uten å måtte prestere. Deretter kan de bruke det de har lært i de andre ulike fagene.

Læringsmål

- Få en bedre forståelse for de grunnleggende begrepene i programmering
- Få bedre kjennskap til Python som programmeringsspråk
- Bli kjent med livekoding som utforskende metode for å lære programmering

Innhold

Lærerveiledning

Livekoding: en metode for å lære vekk programmering

I stedet for å gå gjennom et eksempel på tavla, noe som gjøres hele tiden, kan du som lærer skrive et program sammen med elevene på storskjerm. Kode på direkten.

Forskning (se kilder) sier at livekoding kan være en god metode for å vise elevene hvordan de ulike begrepene og programmering er i praksis. Det blir mer interaktivt, og du kan gå tilbake og endre på koden underveis. Du kan la elevene få delta i oppbyggingen av koden. De kan komme med forslag eller forsøke å forutsi hva som vil skje dersom koden kjøres på ulike tidspunkt. Skulle du som lærer gjøre en feil underveis, er det helt greit. Det vil vise elevene at det er greit om de også gjør feil. Elevene kan også inkluderes i feilsøkingen. Python vil dessuten fortelle hva som er feil slik at det kan rettes opp. Det tar tid å bli god til å programmere, men alle kan lære det.

Figur 1: Skjermbilde fra video fra Udir på Vimeo

Kode en konkret situasjon

Hvordan kan vi vise elevene variabler, løkker, vilkår, funksjoner og importering i en og samme kode? Vi kan gjennom et konkret scenario, en «historie» flette inn bruken av disse kodebegrepene, for å vise hvordan alle disse kan brukes i programmering.

Ta for deg et bestemt scenario, en konkret situasjon og bygg opp en programkode som fletter inn bruken av variabler, løkker, vilkår, funksjoner og importering. Gjør det til en historiefortelling. Det kan være hvilken som helst annen historie, men det er litt engasjerende å finne noe som er enten litt spennende eller noe vi kan kjenne oss igjen i.

Eksempler på ulike situasjoner/historier å livekode:

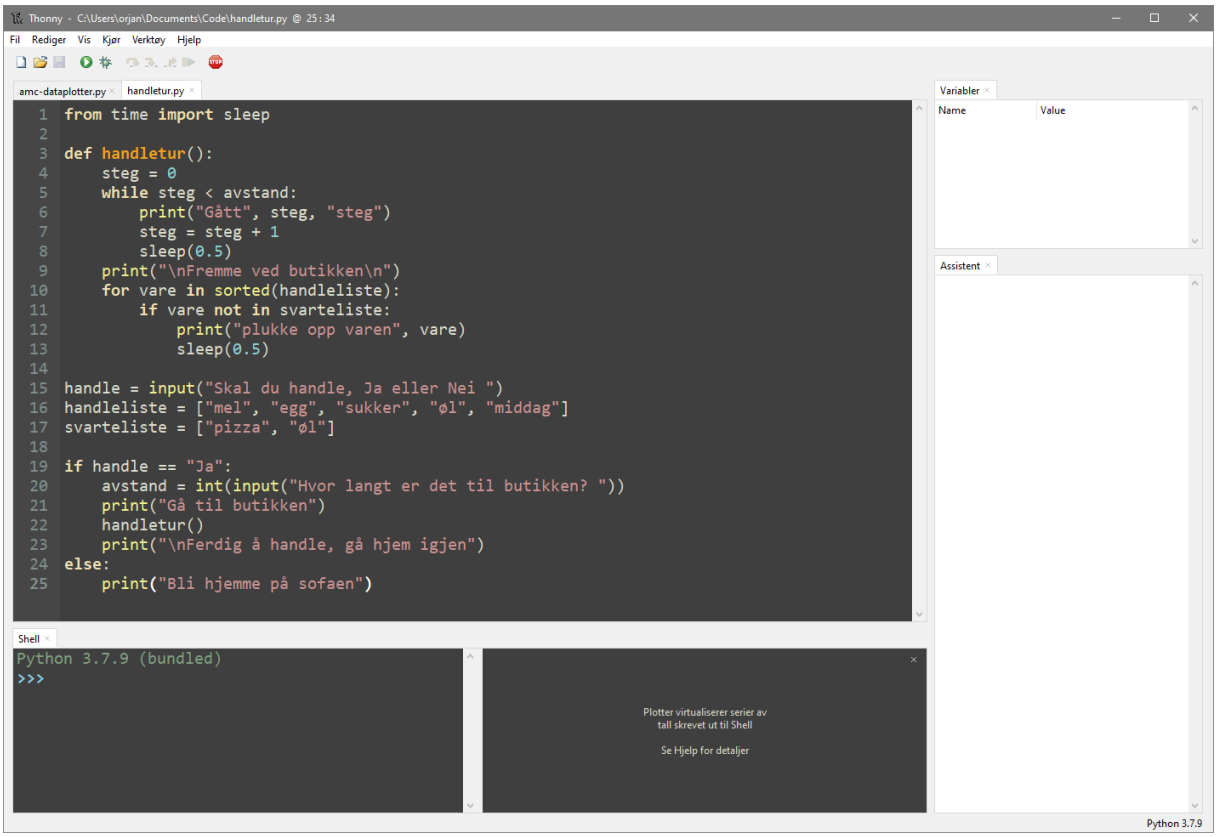
- Handletur på butikken, liste fra pappa eller mamma?
- Space story, redde skolen fra romvesen

Dette er bare et par eksempler på hvordan livekoding kan brukes, men det kan brukes på mange flere områder, egentlig hva som helst. Alle programmeringsøvelser i ulike fag kan benytte denne metodikken, inkludert matematikk. Eksempler kan være:

- Tegne bilder med Python og Turtle-biblioteket
- Arbeide med datasett fra json, csv, xml eller api-er for å visualisere og analysere data
- Øvelser med tall og tekst og språk
- Bygge ting i Minecraft
- Biologiske emner som å transkribere DNA → RNA

Tips til livekoding

- Lær deg en historie godt på forhånd. Det kan virke mot sin hensikt hvis du er usikker på hvordan du skal gå frem.
- Ha en jukselapp med forslag til ferdig kode og stegene du ønsker å gjøre
- Snakk underveis om ting å vurdere. Se eksempel på dette i Aktivitet 1.
- Sett opp et kodeverktøy (for eksempel Thonny) på PC og speile innholdet på PC til storskjerm. (CTRL + L vasker vekk tidligere meldinger i Thonny)
- La elevene skrive koden samtidig på sine maskiner. Ikke bare se på. De lærer av å skrive kode også. Da kan de også utvide koden sin selv underveis.
- Stopp opp med jevne mellomrom og la de komme med forslag
- Kjør koden av og til for å se hva som skjer. Det er helt greit at den ikke fungerer som den skal, det åpner opp for feilsøking.
- Ta deg god tid til dette. Kanskje kan dere fortsette senere, la de reflektere til neste gang hvordan dere kan løse en utfordring i koden.
- Hvis noen elever er erfarne programmerere, kan de utvide koden på egen hånd.
- Det kan være en veldig god strategi å begynne å programmere en historie sammen og overlate til elevene å fullføre historien på egen hånd, «Choose your own adventure» som Kodeskolen kaller det. Enten i mindre grupper eller individuelt. Slik får de også mer eierskap til koden, fremfor å bare skrive det samme som læreren.



Figur 2: Skjermbilde fra kodeverktøyet Thonny

Kort om Algoritmisk tenkning og programmering med Python

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.

Hva er 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 datatyper, om de er tall, bokstaver, en dato eller noe annet. Slike datatyper er også litt typisk for programmeringsspråk å forhåndsdefinere. Altså, du har en bestemt rekke ulike datatyper du kan forholde deg til å bruke i algoritmene.

Kort fortalt med eksempel: Variabler er en verdi tilordnet et ord eller en bokstav:

```
handle = input("Skal du handle, Ja eller Nei ")
handleliste = ["mel", "egg", "sukker", "øl", "middag"]
svarteliste = ["pizza", "øl"]
```

Løkker er en gjentagende hendelse. 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, noe som ville blitt vanskelig å huske for personen som ville til butikken.

Kort fortalt med eksempler: Løkker er en gjentagende hendelse, noe som skal skje en eller flere ganger:

```
while steg < avstand:
    print("Gått", steg, "steg")
    steg = steg + 1
    sleep(0.5)
```

```
for vare in sorted(handleliste):
      if vare not in svarteliste:
            print("plukke opp varen", vare)
            sleep(0.5)
```

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.

Kort fortalt med eksempel: Vilkår er en logisk tenkemåte om å se etter en bestemt situasjon og hvis den inntreffer så skal noe skje:

```
if handle == "Ja":
    avstand = int(input("Hvor langt er det til butikken? "))
    print("Gå til butikken")
    handletur()
    print("\nFerdig å handle, gå hjem igjen")
else:
    print("Bli hjemme på sofaen")
```

Hva er 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.

Funksjoner i programmering fungerer på samme måte som funksjoner i matematikken. Vi har en funksjon som tar noe inn og gir noe annet ut. 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.

```
def handletur():
    steg = 0
    while steg < avstand:
        print("Gått", steg, "steg")
        steg = steg + 1
        sleep(0.5)
    print("\nFremme ved butikken\n")
    for vare in sorted(handleliste):
        if vare not in svarteliste:
            print("plukke opp varen", vare)
            sleep(0.5)
```

Importering av ekstra funksjonalitet i Python

Python og andre programmeringsspråk har en begrenset mengde funksjonalitet innebygd. Det er alt det grunnleggende man kan gjøre, slik som å kunne lage en funksjon, bruke løkker og variabler. Det er også noen innebygde funksjoner i Python, for eksempel *print()* som skriver ut noe til kommandolinjen. Men med en gang du ønsker å gjøre noe mer, trenger du å hente inn mer utenfra. Det gjør vi med importering.

Du kan tenke på importering som å låne en bok på biblioteket. Det kan hende vi bare er interessert i et kapittel fra boken, eller hele boken.

Hvis vi bare ønsker et enkelt kapittel, trenger vi ikke ta med oss hele boken, da river vi ut de sidene fra boken som er interessant og tar de med oss. (Ikke gjør dette på ordentlig, da blir bibliotekaren sur).

Fra boken *time,* røsk med deg kapittelet om *sleep*:

```
from time import sleep
```

Ta med hele boken *time*:

```
import time
```

Når vi har hentet inn ekstra muskler til programmet vårt, kan vi bruke det ved å henvise til kilden. Når vi låner med oss hele boken, må vi peke til boken på denne måten når vi skal bruke et kapittel fra den, slik:

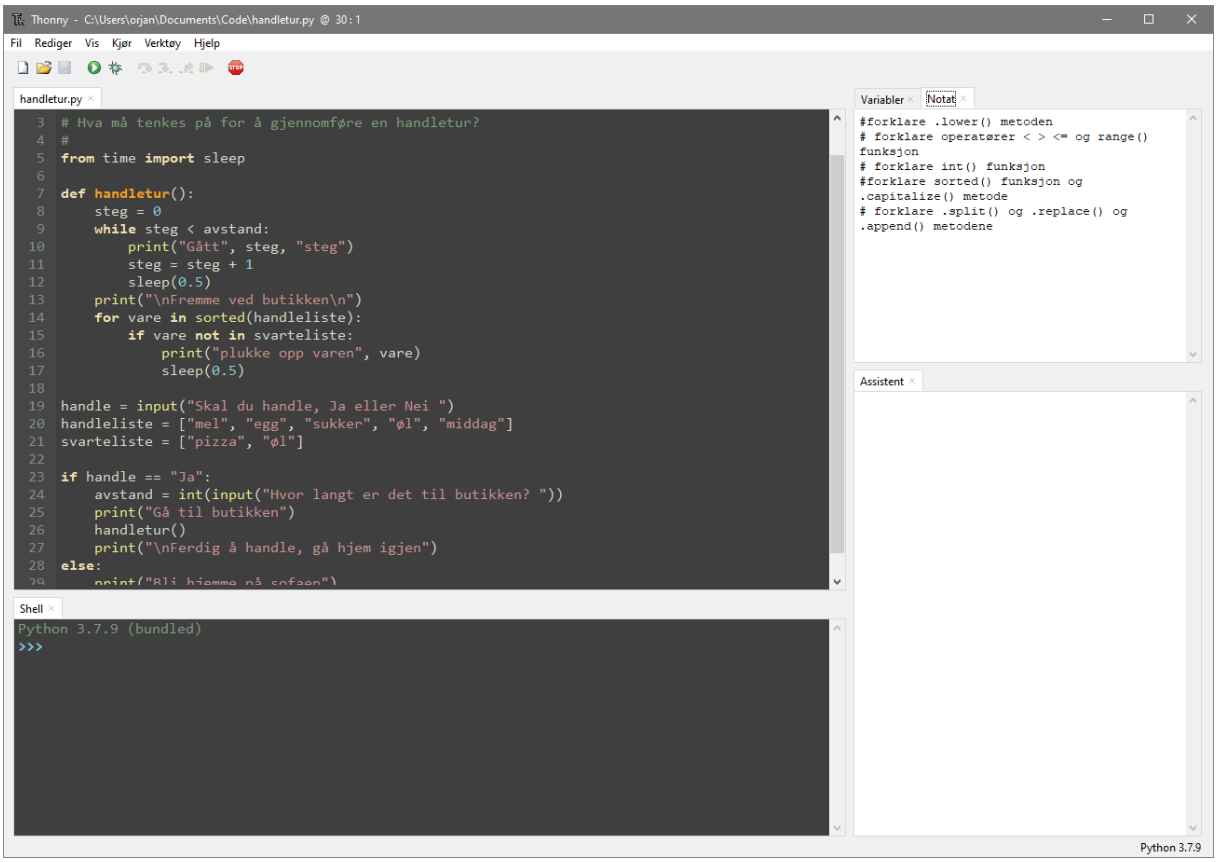
```
time.sleep(4)
```

Har vi derimot røsket ut sidene fra boken, kan vi bare bruke det direkte:

```
sleep(4)
```

Aktiviteter å gjennomføre med elevene

Start opp et kodeverktøy (for eksempel Thonny) og prøv et av kodeoppdragene. Thonny er et enkelt, men glimrende kodeverktøy for å komme i gang med Python-programmering.



Det kan lastes ned på https://thonny.org og kan kjøre på Linux, Windows, Mac og Raspberry Pi.

Kodeoppdrag 1: Handletur på butikken

Her er noen steg for å bygge opp koden sammen med elevene:

1. Ta en beslutning om du skal handle. Fyll en variabel med input()
2. Hva skal vi handle. Sett opp en liste.
3. Sett opp et vilkår, dersom vi skal handle, kjør en handlefunksjon.
4. Lag skjelettet til en funksjon om å handle
5. Ta en beslutning om hvor langt er det til butikken. Fyll en ny variabel med input()
6. Gå til butikken med while-løkke.
7. Vi går litt sakte, vi er jo ikke The Flash. Importering av ekstra funksjonalitet sleep()
8. Vilkår: er vi kommet til butikken, gå gjennom handlelisten med en for-løkke
9. Pynte på koden og litt utvidelser
    a. Sortere handlelisten med innebygd sorted()
    b. Litt penere med mellomrom i teksten som skrives ut, bruk «\n» for newline
    c. Er det varer vi ikke burde kjøpe, lag en svarteliste og sjekk mot den med vilkår

Er det engasjerte og erfarne elever, kan du be de utvide koden og gjøre den omfattende, med for eksempel:

- å hente listen fra en tekstfil eller enkel database
- gjøre koden til en nettapplikasjon med Flask
- Lage et grafisk grensesnitt med Tkinter
- utvide til å passe med valgfri butikk. Endre sorteringen, slik at handleturen er mest mulig effektiv
- Legge inn forventet pris på varene, bruke annen type en liste til å holde på handlelisten, for eksempel dict-typen
- Priser på varer
- Er det øl, spør om alder

Kodeoppdrag 2: Angrep fra rommet

Det er et romskip over skolen og vi må samles i biblioteket for å legge en forsvarsplan.

Ting å vurdere til koden:

- Input: Hvem er med? Hvor langt er det til biblioteket?
- Funksjon: Dra til biblioteket
- Variabler: Liste over elever som er med, liste over ting å gjøre, tiden går
    - Sjekk om listen har store forbokstaver i navnene, endre case med upper() og lower()
- Beslutninger: Skal vi gjøre dette? Er alle her?
- Løkker: Gå til biblioteket

Vedlegg 1 – eksempel på handletur.py

```
# Kodeoppdrag 1: En handletur på butikken
#
# Hva må tenkes på for å gjennomføre en handletur?
#
from time import sleep

def handletur():
    steg = 0
    while steg < avstand:
        print("Gått", steg, "steg")
        steg = steg + 1
        sleep(0.5)
    print("\nFremme ved butikken\n")
    for vare in sorted(handleliste):
        if vare not in svarteliste:
            print("plukke opp varen", vare)
            sleep(0.5)

handle = input("Skal du handle, Ja eller Nei ")
handleliste = ["mel", "egg", "sukker", "øl", "middag"]
svarteliste = ["pizza", "øl"]

if handle == "Ja":
    avstand = int(input("Hvor langt er det til butikken? "))
    print("Gå til butikken")
    handletur()
    print("\nFerdig å handle, gå hjem igjen")
else:
    print("Bli hjemme på sofaen")
```

Vedlegg 2 – eksempel på romvesen.py

```
# Kodeoppdrag 2: Angrep fra rommet - et spill?
#
# Det er et romskip over skolen og vi må samles i biblioteket for å legge en
forsvarsplan.
#
from time import sleep

tid_til_angrep = 10
hvem_er_med = []
ditt_navn = ""

print("Det har blitt observert et romskip over skolen. Vi må samles i biblioteket
for å legge en plan.\n")

ditt_navn = input("Hva heter du? ")
hvem_er_med = input("Hvem andre er med? ").replace(" ", "").split(",")
hvem_er_med.append(ditt_navn)
print("\nFlott, da har vi med oss {} stykker, kjempebra
{}!\n".format(len(hvem_er_med), ditt_navn.capitalize()))

for navn in sorted(hvem_er_med):
    print(navn.capitalize())

print("\nRomvesene angriper om {} minutter\n".format(tid_til_angrep))

tid_til_biblioteket = int(input("Hvor mange minutter tar det å komme seg til
biblioteket? "))

tid = tid_til_angrep-tid_til_biblioteket

if tid >= 5:
    print("\nDa har vi {} minutter på oss, ingen tid å miste!\n".format(tid))
    for minutt in range(1,tid+1,1):
        sleep(1)
        print("Det har nå gått {} minutt".format(minutt))

elif tid > 0 < 5:
    print("\nDa har vi bare {} minutter på oss. Vi må virkelig skynde
oss!\n".format(tid))
    for minutt in range(1,tid+1,1):
        sleep(1)
        print("Det har nå gått {} minutt".format(minutt))

elif tid < 1:
    print("\nDa har vi ingen sjanse, vi rekker det ikke. Lykke til!".format(tid))

print("\nFremme ved biblioteket. Da kan vi begynne planleggingen vår. Vi må legge
en strategi for hvordan vi skal beskytte skolen vår.\n")
med_eller_ikke = input("Er du med {}? (Svar ja eller nei)
".format(ditt_navn.capitalize()))

if med_eller_ikke.lower() == "ja":
```

```
    print("\nVeldig bra {}, da setter vi i gang, takk for denne
gangen...\n".format(ditt_navn.capitalize()))
if med_eller_ikke.lower() == "nei":
    print("\nDet er forståelig {}, takk for denne gangen...og lykke
til!\n".format(ditt_navn.capitalize()))
```

Kilder og lenker:

- «Algoritmisk tenkning» 27.03.2019, https://www.udir.no/kvalitet-og-kompetanse/profesjonsfaglig-digital-kompetanse/algoritmisk-tenkning/
- «Programmering i skolen» 17.08.2018, https://vimeo.com/285474709
- «Kompetansepakke: Programmering og algoritmisk tenkning», https://bibsys.instructure.com/courses/387?lang=nb
- «Ten quick tips for teaching programming» 05.04.2018, https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1006023.
- «The effectiveness of live-coding to teach introductory programming» 06.03.2013, https://dl.acm.org/doi/10.1145/2445196.2445388.
- Eksempelkode på GitLab, https://gitlab.com/andoyaspaceeducation/a-kode-sammen-livekoding
- Kodeverktøyet Thonny – thonny.org
- Kodeskolen, forskningslaboratoriet Simula, https://simulakodeskolen.no