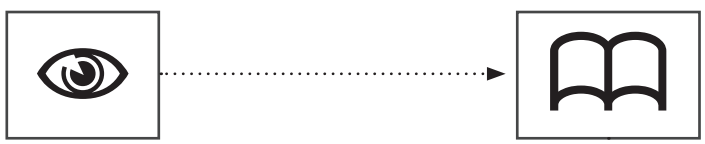
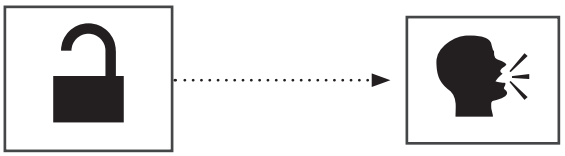
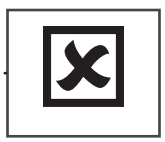
COMPUTATIONEL TANKEGANG I TEKNOLOGIFORSTÅELSESFAGET



Af **Ole Caprani og Line Have Musaeus**



Denne artikel indeholder en kort beskrivelse af hvordan begrebet Computationel Tankegang (CT) har udviklet sig og hvordan centrale elementer af CT kan beskrives som modellering ved hjælp af algoritmisk tankegang og automatisering. Herudover gennemgår artiklen, hvordan CT-begrebet kan operationaliseres i konkrete aktiviteter med elever. Dette gøres i form af en udførlig beskrivelse af to læringsaktiviteter.



Indledning
Denne artikel præsenterer en konkretisering af kompetenceområdet 'Computationel Tankegang' (CT) i Teknologiforståelsesfaget (UVM, u.å.) med udgangspunkt i den store mængde af forskning der efterhånden findes om CT i uddannelse. Kernen i artiklen er et forsøg på at operationalisere CT som begreb, så de centrale elementer, som indgår i beskrivelsen af CT i læseplanen, kan være udgangspunkt for en praksis i undervisningen.

Operationaliseringen sker i to trin:

» For det første giver vi vores bud på, hvad der er de centrale elementer i CT i læseplanen, når CT skal udmøntes som undervisning. De centrale elementer er *algoritmer, data* og *automatisering gennem programmering*. Vores bud baseres dels på oprindelsen af begrebet CT, dels på dele af de omfattende diskussioner, som indførelsen af CT i uddannelse har afstedkommet.

» For det andet indeholder artiklen konkrete forslag til, hvordan CT undervisning kan finde sted i grundskolen.

Computationel tankegang - historisk set
Umiddelbart skulle man tro, at begrebet CT knytter sig til digitale computere, men f.eks. algoritmebegrebet, der i dag opfattes som centralt i CT, har været anvendt i århundreder (Denning & Tedre, 2018; Curzon et al., 2019).

Seymour Papert var den første til at bruge begrebet Computational Thinking (CT) i sin bog Mindstorms fra 1980 (Papert, 1980, s. 182). Samtidig er Papert den første til at lave en omfattende beskrivelse af, hvordan computere kan indgå i en CT-læring i skolen. I Mindstorms beskriver Papert, hvordan elever kan arbejde med elementer af CT ved at instruere en tegnerobot til at tegne geometriske figurer på et stykke papir, f.eks. en firkant. For at få en ide om, hvilke elementer af CT Papert beskriver og hvordan disse elementer også indgår i CT i læseplanen, beskriver vi lidt mere detaljeret, hvordan elevernes arbejde med tegnerobotter, som beskrevet af Papert, berører CT-begreber fra læseplanen (begreberne er vist med kursiv). I bogen beskriver Papert, hvordan eleverne kan arbejde med en tegnerobot. Eleverne kan instruere tegnerobotten ved at give tegnerobotten kommandoer som "drej 90 grader til højre" eller "kør fremad 20 cm". For at få tegnerobotten til at tegne en firkant skal eleverne altså finde ud af, hvilken række eller *sekvens* af kommandoer, som får tegnerobotten til at tegne en firkant. En sådan sekvens af kommandoer, f.eks. "kør 20 cm", "drej 90 grader", "kør 20 cm", "drej 90 grader", kaldes en *algoritme*. En algoritme er en beskrivelse af en fremgangsmåde – i dette tilfælde for tegnerobotten – som trin for trin fører til et ønsket resultat – i dette tilfælde en tegning af en firkant. Når eleverne har fundet frem til en sådan sekvens af kommandoer, altså en algoritme, kan eleverne beskrive algoritmen for tegnerobotten som et *program*, en sekvens af kommandoer skrevet i et *programmeringssprog*, som robotten kan forstå og udføre. F.eks. vil kommandoerne i programmeringssproget LOGO være skrevet som FORWARD 20, TURNRIGHT 90. Dernæst vil robotten udføre programmet og programmet styrer dermed robotten, så den på egen hånd tegner en firkant.

Ifølge Papert lærer eleverne at programmere en tegnerobot ved at eksperimentere med og undersøge kommandoer, algoritmer og programmer, men eleverne lærer desuden matematikbegreber gennem programmeringen. Så det at lære programmering bliver også et værktøj til at lære begreber fra andre fagligheder som f.eks. matematik. Curzon et al. (2019) forklarer, at Papert i sin forskning omkring computere og læring i skolen ikke alene er den første til at beskrive ideen om "a new approach to teaching mathematics based on computational methods", men at Papert er også en af de første til at brede det ud til ideen om, at "computational thinking is a way of doing other subjects differently". Denne ide kom til udtryk i en vision om brugen af computere i skolen allerede hos Papert & Solomon i 1972, hvor de igen-

nem en række eksempelprojekter beskriver, hvordan elever ved at arbejde konkret med sådanne projekter kan lære at programmere computere, og altså ikke blot at programmere en tegnerobot. Samtidig vælger Papert & Solomon meget forskellige projekter, som f.eks. at få computere til at spille musik, at skabe tilfældige sproglige sætninger ud fra ordlister i forskellige ordklasser eller at træne en person i additionsstykker. Gennem at udforme algoritmer og programmer i sådanne projekter er ideen, at eleverne også lærer om begreber i musik, sprog og matematik.

I bogen vælger Papert en konstruktivistisk tilgang til, hvordan tegnerobotten kan indgå i elevernes læring af kommandoer, algoritmer og programmer. Denne tilgang er grundlagt af Jean Piaget og tager udgangspunkt i, at elever konstruerer viden ud fra allerede erhvervede erfaringer (Piaget, 1971). Konstruktivisme lægger vægt på, at eleverne erhverver erfaringer ved på egen hånd at arbejde med stoffet og Papert viser, hvordan denne læringsteori kan benyttes til at tilrettelægge aktiviteter, hvor elever arbejder på egen hånd med kommandoer, algoritmer og programmer til tegnerobotten. Bl.a. beskriver Papert, hvordan eleverne kan bruge deres krop til gennem konkrete eksperimenter at forstå, hvordan de enkelte kommandoer styrer tegnerobotten.

Computationel tankegang – som begreb
Jeannette Wing re-introducerede begrebet Computational Thinking i 2006 (Wing, 2006) og gav i 2010 denne forklaring af begrebet:

Computational thinking is the thought processes involved in formulating a problem and expressing its solution(s) in such a way that a computer — human or machine — can effectively carry it out.
Cuny et al., 2010

CT er altså måder at tænke på, når et problem skal formuleres og en løsningsmetode skal beskrives i en sådan form, at en person eller en computer kan følge beskrivelsen og løse problemet.

Jeannette Wings beskrivelser af CT har givet anledning til en mangfoldighed af forskellige udlægninger af CT (Curzon et al., 2019). Fælles for disse beskrivelser er, at CT bliver opfattet meget bredt som måder at tænke på, der bl.a. omfatter evnen til modellering af virkeligheden, strukturering af data, algoritmisk tankegang, abstraktion, generalisering, dekomponering, mønstergenkendelse, logik, automatisering og fejlfinding.

Mark Guzdial (2008; 2019) argumenterer for, at modellering ved hjælp af algoritmisk tankegang og automatisering, forstået som det at formulere en algoritme og omforme den til et program, er specifikt for det datalogiske vidensområde, mens andre begreber fra ovenstående liste, som f.eks. abstraktion, kan genfindes i bl.a. naturvidenskabelig, humanistisk eller psykologisk tankegang (Caspersen et al., 2017).

I forbindelse med at diskutere, hvad CT skal være i undervisning, konstaterer Mark Guzdial (2019), at "Computation *can* play an important role in learning decomposition and abstraction, but those skills don't belong uniquely to computation, or to a class on computational thinking" og derefter stiller han spørgsmålene "What is unique about computation?" og "How small can we make computational thinking?"

Ved at stille sådanne spørgsmål når Guzdial frem til, at CT kan karakteriseres som evnen til at udforme *algoritmer*, evnen til at identificere de *data*, som algoritmen skal bearbejde og evnen til at automatisere en algoritme og tilhørende data til et *program*, som en computer kan forstå og udføre (Guzdial, 2019). Bemærk, at i læseplanen hører algoritme og data til i læseplanens beskrivelse af CT, og programmering er en del af kompetenceområdet 'Teknologisk handleevne'.

Computationel tankegang – i undervisning
I denne artikel er det Guzdials tilgang til CT, som er udgangspunktet for forfatternes forslag til CT i undervisning, idet de beskrevne aktiviteter overordnet fokuserer på model-

lering ved at træne algoritmeforståelse og arbejdet med data. Samtidig udtrykker denne tilgang til CT de evner, som eleverne *i hvert fald* skal have, og som samtidig skaber bro til kompetenceområderne 'Teknologisk handleevne' ved arbejdet med programmering og til 'Digital myndiggørelse' ved arbejdet med modellering og redesign af algoritmer. Denne tilgang foldes ud i de følgende afsnit "Algoritmejagt" og "Computeren spiller musik".

Didaktik
Undersøgelser helt tilbage fra 1980'erne (Guzdial, 2008) har vist, at ikke-programmeringskyndige har svært ved sprogligt at beskrive en fremgangsmåde; f.eks. at beskrive, hvordan en simpel beregningsproces skal udtrykkes præcist, så beregningen kan udføres ved bogstaveligt at følge beskrivelsen af fremgangsmåden.

Lad os tage et lidt gammeldags eksempel: De fleste ved, hvordan de i en konkret situation skal give penge tilbage, men det har vist sig, at det er endog meget svært for ikke-programmeringskyndige at udforme en beskrivelse af en generel fremgangsmåde, altså en algoritme, som i en række trin instruerer en person til at kunne finde frem til de sedler og mønter, som skal gives tilbage for enhver varepris og ethvert betalt beløb. Prøv selv at udforme en sådan algoritmisk løsning, som løser problemet om tilbagebetaling for alle mulige beløb.

Når eleverne skal tilegne sig algoritmisk tankegang som en del af CT, er målet, at eleverne selv skal kunne udforme algoritmer som præcise, sproglige beskrivelser af fremgangsmåder. Eleverne skal altså selv kunne lave algoritmiske beskrivelser. I lyset af de netop omtalte erfaringer fra tidligere tiders forskning skal elevernes arbejde med algoritmer, algoritmeforståelse, algoritmebeskrivelser altså tilrettelægges didaktisk, så de kan nå dette mål.

To didaktiske principper: 'tinkering' og 'use-modify-create'
Den konkrete tilrettelæggelse af aktiviteterne (beskrevet nedenfor) "Algoritmejagt" og "Computeren spiller musik" er inspireret af de to didaktiske principper 'use-modify-create' (Lee et al., 2011) og 'tinkering' (Resnick & Rosenbaum, 2013). Principperne har med succes været brugt i gymnasieundervisning i Danmark (Musaeus & Musaeus, 2019).

Use-modify-create
Det første didaktiske princip lægger vægt på elevernes egne eksperimenter og undersøgelser som i konstruktionisme og lader først eleverne *bruge* (use) pre-designede algoritmer og programmer, for derefter at opfordre eleverne til at *ændre* (modify) og til sidst *skabe* (create) nye dele af en algoritme eller et program. Herved opnås både en progression og en iteration i undervisningsaktiviteterne.

Tinkering
Dette andet didaktiske princip indebærer en legende, eksperimenterende og undersøgende tilgang til det fænomen, der arbejdes med. Når f.eks. en algoritmisk beskrivelse skal omformes til et program, vil en brug af dette princip give eleverne tid til at lege, eksperimentere og undersøge de mange mulige valg i denne proces.

De følgende konkrete forslag til undervisning i CT er struktureret ud fra disse principper. I afsnittet "Algoritmejagt" bruges 'use-modify-create', da eleverne f.eks. selv finder algoritmiske beskrivelser fra deres hverdag, dernæst *bruger* eleverne algoritmerne i to spil, vendespil og Sorteper. Herefter *ændrer* eleverne på beskrivelser f.eks. i Sorteper og endelig opfordres de til selv at udforme, *skabe*, algoritmer f.eks. for et spil, de kender eller et spil, de finder på. I afsnittet "Computeren spiller musik" bruges bl.a. 'tinkering', idet eleverne opfordres til at lege, undersøge og eksperimentere med instrumentvalg, tempoændringer under musikafspilningsprocessen og udformning af rytmiske mønstre. Og måske lege med musikalsk samspil mellem flere computere.

To CT-aktiviteter i undervisning
Resten af denne artikel beskriver to forslag til CT-aktiviteter, nemlig "Algoritmejagt" og "Computeren spiller musik". Gennem disse og lignende aktiviteter er det meningen, at eleverne skal opnå en evne til at kunne lave algoritmiske beskrivelser og identificere de data, som skal indgå i algoritmens databehandling. I beskrivelsen af de to aktiviteter kursiveres de begreber, som står under kompetenceområderne CT og 'Teknologisk handleevne' i læseplanen for Teknologiforståelsesfaget (UVM, u.å.).

Aktivitet 1: Algoritmejagt
Vi er omgivet af *algoritmiske beskrivelser*, der forklarer, hvordan vi f.eks. får en parkeringskvittering ud af en parkeringsautomat, betaler i en automat med MobilePay/Dankort eller registrerer og betaler varer ved en selvbetjeningsautomat i et supermarked.

Ideen i denne aktivitet er at få eleverne til at gå på jagt efter og identificere sådanne algoritmiske beskrivelser; fx madopskrifter, samleinstruktioner for LEGO, skattekort og rutebeskrivelse fra Google Maps. Start aktiviteten med at vise eleverne eksempler på algoritmiske beskrivelser og lad dem så finde lignende beskrivelser. Ud fra de fundne beskrivelser er det så meningen, at eleverne skal danne en første forståelse af, hvad en *algoritme* er.

Denne forståelse kan f.eks. opnås ved, at eleverne for hver algoritmisk beskrivelse skal afgøre, om de enkelte trin er beskrevet, så en person kan forstå og udføre dem. Eleverne kan også prøve at finde ud af, om resultatet er det samme hver gang, en algoritmisk beskrivelse følges, f.eks. af to forskellige personer. På denne måde vil eleverne opnå en forståelse af det, som er *karakteristisk ved en algoritme*, som en person skal kunne følge:

» En algoritme er en fremgangsmåde som kan forstås og følges af en person, og følges algoritmen, fører det til et ønsket resultat.

» En algoritme skal være beskrevet, så de enkelte trin præcist og entydigt instruerer en person, som følger algoritmen.

» Forskellige personer skal komme frem til samme resultat, hvis de følger den samme algoritme.

Eleverne vil også opdage, at nogle algoritmiske beskrivelser udelukkende består af tekst, men at der ofte indgår grafiske elementer og billeder. Algoritmiske beskrivelser kan have tegneserieform eller være en instruktionsvideo. At der findes så forskellige udtryksformer skyldes f.eks., at en beskrivelse af fremgangsmåden til at binde en knude er næsten umulig at lave i tekst, men forholdsvis nemt at lave i en instruktionsvideo.

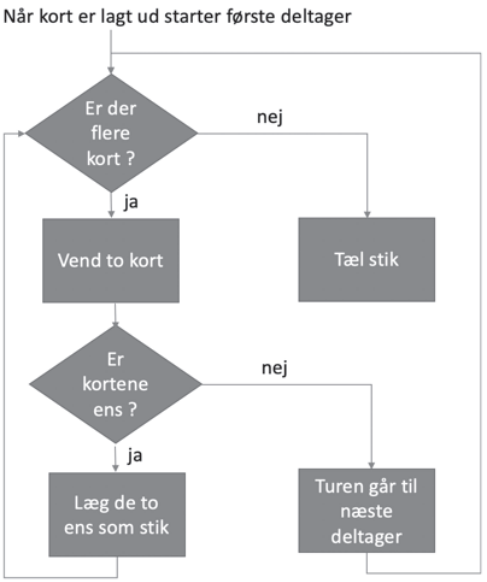
Et fællestræk i sådanne hverdagsbeskrivelser af fremgangsmåder er, at *rækkefølgen* af, hvornår de enkelte handlinger skal udføres, er tydeliggjort ved f.eks., som det ses på billedet, at nummerere de enkelte trin. Det tidslige forløb af handlingerne er altså fastlagt i beskrivelsen som en fast sekvens af handlinger, 1, 2, 3, osv. Der er en simpel en-til-en korrespondance mellem det skrevne og de udførte handlinger.

I andre algoritmiske beskrivelser er der *ikke* en sådan simpel korrespondance. Tænk på to eller flere personer, som spiller vendespil. Vendespil spilles jo ved, at deltagerne følger et sæt regler:

Spilleregler for vendespil

Alle kort lægges ud på bordet med bagsiden opad.

Den første deltager vender to kort, er de forskellige, går turen til den næste deltager. Vender denne deltager 2 ens kort, er det et stik, der lægges ved den heldige deltager, som må vende to nye kort, før turen går videre. Når der ikke er flere kort, tæller deltagerne deres stik, og den der har flest stik, har vundet.



Efter at kortene er lagt på bordet i det første trin, folder spillet sig ud ved, at deltagerne følger reglerne i det andet trin. Den algoritmiske beskrivelse i det andet trin er imidlertid ikke en fast sekvens af handlinger, deltagerne skal udføre, for handlingerne, de enkelte deltagere udfører, *afhænger af situationen* undervejs i spillet: "Vender denne deltager to ens kort...". Når det er en deltagers tur, kan deltageren altså skulle vende to kort enten én eller flere gange, inden turen går videre til den næste spiller eller til spillet er slut.

Reglerne for vendespil er altså en algoritmisk beskrivelse, som fastlægger *alle de mulige tidslige forløb* af deltagerhandlinger i de enkelte mulige spil, som er resultatet af, at deltagerne følger algoritmen i reglerne.

En måde at tydeliggøre de mulige forløb af handlinger i et spil på er at lave et *rutediagram*, der ved hjælp af kasser (med handlinger), romber (med spørgsmål) og pile (med retning) imellem kasser og romber, visualiserer rækkefølgen, de enkelte trin i algoritmen udføres i. De mulige ruter gennem et rutediagram er så de mulige resultater af at udføre algoritmen, altså de mulige veje, et vendespil kan tage:

Prøv at lade eleverne spille vendespil efter rutediagrammet og få eleverne til at øve sig i at følge reglerne slavisk, ikke sådan som de er vant til at spille vendespil, men ved præcist at følge rutediagrammets beskrivelse af vendespil. En af eleverne kan have rollen som regelholder, som følger algoritmen og fortæller deltagerne, hvad de skal gøre.

Afslut evt. aktiviteten med at lade eleverne reflektere over rutediagrammets begrænsninger. Opstod der usikkerhed blandt spillerne undervejs? Hvorfor? Hvordan kan rutediagrammet ændres, så disse usikkerheder kan undgås? Herved arbejder eleverne med det didaktiske princip 'use-modify-create', og med områder, der omhandler "at kunne vurdere muligheder og begrænsninger i de computationelle tankegange", som beskrevet i læseplanen.

Rutediagrammet kan være udgangspunkt for at forstå, hvordan en computer kan programmeres til i et vendespil først at lægge kortene tilfældigt ud på en skærm, dernæst at holde øje med stik og endelig afgøre, hvornår et spil er færdigt. Lad eleverne spille et sådant vendespil på en computer og lad

eleverne sammenholde spillets gang på computeren med ruterne i rutediagrammet.

Prøv også at lade eleverne spille fx Sorteper:

Spilleregler for Sorteper

Kortgivning:
Alle kortene deles ud. Det spiller ingen rolle, hvis nogen får flere kort end andre.

Spillets gang:
Sorter kortene på hånden i par f.eks. to femmere, to damer, to tiere osv. Husk! Spar knægt kan naturligvis ikke tages med i et par!

De kort, som danner par skal vises for medspillerne og lægges ned på bordet.

Kortgiver lader medspilleren til venstre for sig trække et kort fra sin hånd. Kan han/hun bruge det til at danne et par, så lægges de ned. Hvis ikke, så beholder han/hun kortet.

Derefter lader han/hun sin sidemand trække et kort fra sin hånd. Sådan fortsætter man bordet rundt.

Spillet slutter, når der kun er et kort tilbage. Den som sidder med det sidste kort er Sorteper.

Husk at minde eleverne om, at de udelukkende skal spille spillet efter reglerne, altså at reglerne skal følges bogstaveligt. Eleverne skal ikke spille ud fra, hvad de tror, reglerne er. Undervejs skal eleverne finde ud af, om reglerne kan forstås, som de står og om der er mangler i reglerne, f.eks. hvad skal en spiller gøre, når spilleren ikke har flere kort på hånden?

For de algoritmiske beskrivelser, som eleverne dernæst selv finder eller for spillet Sorteper, kan eleverne forsøge at omforme beskrivelserne til rutediagrammer. Under arbejdet med dette vil eleverne formentlig blive klar over, at en algoritme bl.a. består af beskrivelser:

» af *handlinger*

» af *rækkefølge af handlinger*

» af *valgsituationer*, hvor svar på spørgsmål afgør næste handling i algoritmen

» af *gentagelser* af handlinger og valgsituationer

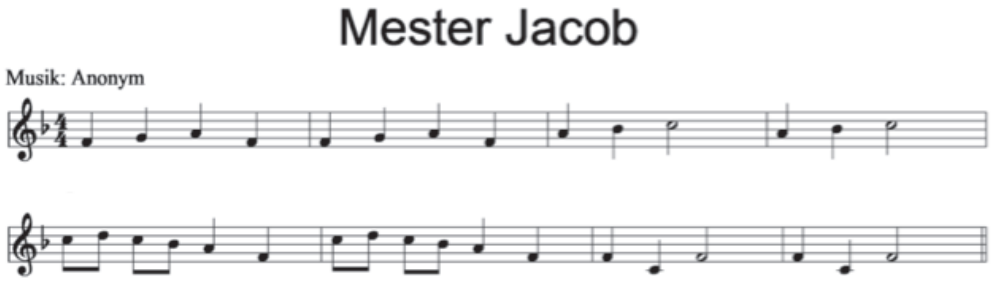
For at indse, at disse elementer ofte indgår i algoritmiske beskrivelser, kan eleverne finde de sproglige udtryk, som benyttes til at angive handlinger, rækkefølge af handlinger, valgsituationer og gentagelser i de algoritmiske beskrivelser, de allerede har fundet.

Afslut evt. aktiviteterne med at lade eleverne reflektere over den overordnede struktur af de algoritmiske beskrivelser, rutediagrammerne. Hvilke dele består de af? Hvordan ser gentagelser ud i rutediagrammer?

Aktivitet 2: Computeren spiller musik
Nogle algoritmebeskrivelser er så præcise, at algoritmen umiddelbart kan *automatiseres* ved at udforme algoritmen som et tilsvarende *program* til en computer, der så kan forstå og udføre algoritmen repræsenteret som et program. Det kan f.eks. være en beskrivelse af en fremgangsmåde såsom noder for et stykke musik.

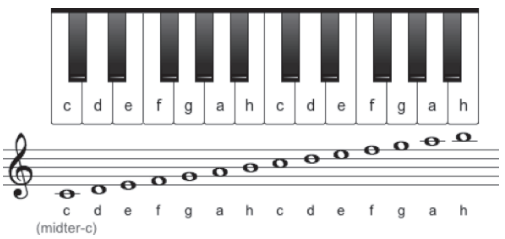
Idéen i denne aktivitet er, med udgangspunkt i noderne for Mester Jakob, at lave et program som kan spille Mester Jakob på en computer. Under omformningen af noderne til brug i et computerprogram ses på forskellige måder tal benyttet som data.

Automatisering af nodeafspilning
Udgangspunktet er altså noderne for Mester Jacob:



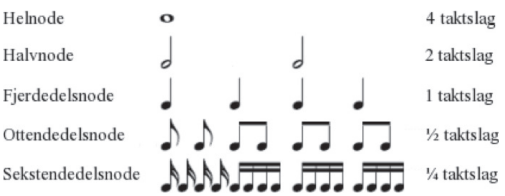
Noderne til Mester Jacob (https://www.aarhussymfoni.dk/harmoni/)

I noderne vises, hvilke toner der skal spilles, rækkefølgen de skal spilles i og varigheden af de enkelte toner. Nodehovedets placering på nodestregerne fortæller, hvilken tone som skal spilles, f.eks. defineret ud fra tangenterne på et klaver:



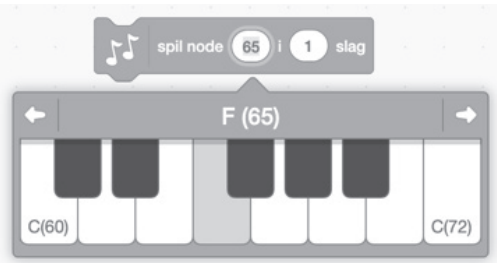
Mester Jacob starter altså med de fire toner f, g, a og f.

Varigheden af de enkelte toner målt i taktslag er bestemt af nodernes udseende:



I Mester Jacob er der altså toner, som varer 2 taktslag, 1 taktslag og ½ taktslag.

I blokprogrammeringsværktøjet Scratch, udvidet med et musikbibliotek, findes en blok som kan afspille en enkelt tone, f.eks. den første tone i Mester Jacob, tonen f:

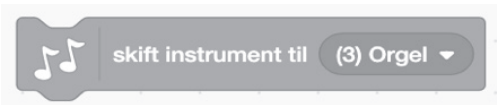


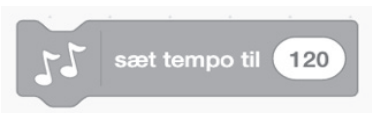
I blokken "spil node" repræsenteres tone og varighed som to tal, hhv. 65 og 1. De to tal er de *data*, som skal angives i blokken for at fortælle blokken den konkrete tone, som blokken skal spille, samt tonens varighed. De fire første toner i Mester Jacob kan programmeres med fire blokke:

På lignende måde kan vi gå videre og omsætte alle noderne i Mester Jacob til 32 blokke. Nu er der imidlertid nogle *gentagelser* i Mester Jacob, som i programmeringen kan udnyttes til at gøre programmet kortere. De fire første toner skal f.eks. gentages to gange:

Lad eleverne afspille disse otte toner, og lad derefter eleverne skrive resten af programmet ud fra noderne. Derved bevæger aktiviteterne sig igennem de første dele af det didaktiske princip om 'use-modify-create'.

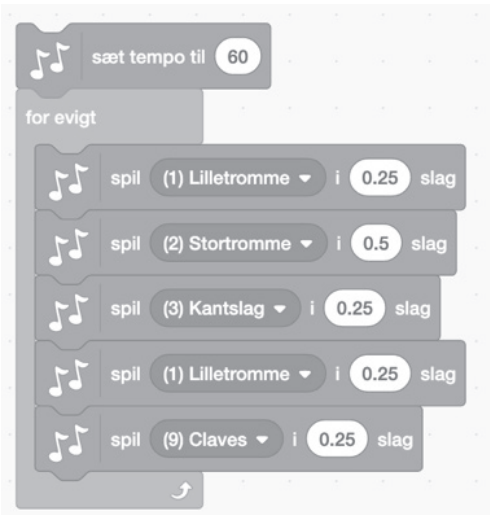
Brug det didaktiske princip om 'tinkering' og lad også eleverne eksperimentere med forskellige instrumenter ved i starten af programmet at bruge blokkene:





I noderne for Mester Jacob er det jo ikke specificeret, hvilket instrument som skal benyttes. I "skift instrument til"-blokken er *data* instrumentnummeret, her 3, som står for orgel. Der kan endda skiftes instrument midt i melodien. En anden oplysning som ikke står i noderne er, hvilket tempo der skal benyttes ved afspilningen. I blokken "sæt tempo til" er tallet 120 *data*, og tallet angiver tempoet i enheden beats per minute (bpm). Tempo 120 er 2 taktslag per sekund, altså en varighed på 0,5 sekunder imellem taktslag. Husk, tempoet kan ændres undervejs.

Slut eventuelt med at bruge trommelydene til at lave rytmiske mønstre i Scratch, f.eks.:



Lad eleverne ændre i valg af trommelyde, ændre varighed af taktslagene, ændre tempo, osv. Gennem aktiviteten lærer eleverne måske også noget om musik- og matematikbegreber (decimalbrøker).

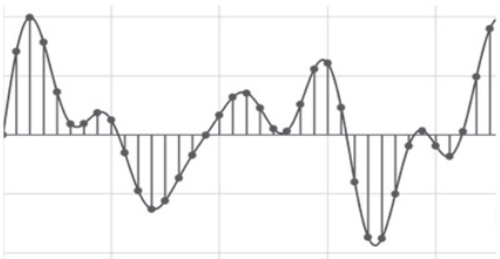
At lege og eksperimentere med sådanne rytmiske mønstre ligner Paperts forslag om, at elever ved på egen hånd at arbejde med en tegnerobot dels lærer om kommandoer, algoritmer og programmer, dels lærer om matematikbegreber.

Arbejdet med trommelyde i rytmiske mønstre giver samtidig eleverne mulighed for at forstå, at algoritmer og programmer *ikke kun udformes for at løse problemer.* Programmering af i Scratch kan også bruges til, at

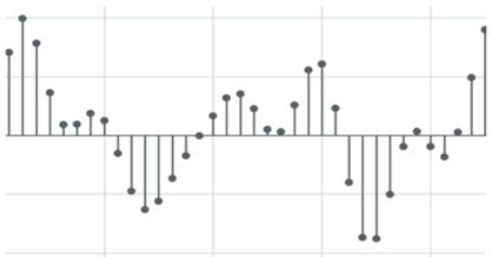
eleverne *udtrykker sig musikalsk,* og tegnerobotter kan gennem programmering ikke alene bruges til at tegne firkanter, men tegnerobotter kan desuden bruges til at *skabe krusedullekunst* (Caprani, 2015).

Musik som data

Når computeren eller mobiltelefonen spiller musik, f.eks. streamet fra Spotify, er musikken **ikke** repræsenteret som i Scratch. I stedet er musikken repræsenteret med udgangspunkt i de fysiske svingninger som opstår omkring instrumenter og sangere, når de skaber lyd. Når musikken optages med en mikrofon, omsættes de fysiske svinger i luften til elektriske svingninger, illustreret med nedenstående kurve:



Hvis mikrofonen er tilsluttet en computeren kan computeren ud fra de elektriske svingninger omsætte kurven til tal, idet computeren måler størrelsen af udsvingene i en række punkter på kurven:



Musikken i computeren er altså repræsenteret som en række tal, altså *data*, svarende til længden af stregerne fra punkterne til den vandrette streg i figuren ovenfor. Lyden som fysiske fænomen *modelleres* altså ved denne repræsentation som digitale data, der dernæst lagres på en computer i et specielt filformat. Når musik lagret i en fil skal spilles på en højtaler, der er tilsluttet en computer, vil en algoritme i computeren benytte data (højderne) til at sætte højtaleren i svingninger svarende til punkterne. Hvis der er punkter nok, vil højtaleren få luften til at svinge som kurven i figuren.

Frembringelsen af musik ved hjælp af computeren er altså beskrevet meget forskelligt i et Scratch-program og i en fil, men fælles er det, at *data i form af tal* repræsenterer musikken i computeren.

Afslutning

Gennem arbejdet med aktiviteter som de foreslåede i denne artikel kan den computationelle tankegang opøves hos eleverne. De opnår en indsigt i algoritmiske tankemåder, en forståelse for digitale data og for modellering. Dette udvikler kompetencer, som er essentielle i computationel tankegang, men også kompetencer indenfor andre af Teknologiforståelsesfagets områder. Kompetencer indenfor 'Teknologisk handleevne' øves f.eks., når eleverne skal skrive et program færdigt ud fra noder i aktiviteten "Computeren spiller musik". I 'Digital myndiggørelse' er kompetencer som kritisk, refleksiv og konstruktiv undersøgelse og forståelse centrale. Disse øves f.eks. gennem aktiviteten "Algoritmejagt", hvor eleverne skal afprøve og reflektere over algoritmernes rækkevidde ved dels at vurdere et rutediagrams muligheder og begrænsninger, dels ved måske gennem jagten på algoritmer at finde ud af, hvordan algoritmer og data indgår i hverdagens digitale artefakter som styring af reklamer i Google, nyhedsstrømme i Facebook, mm.

Forhåbentlig er eleverne blevet CT-nysgerrige i forhold til hverdagens digitale artefakter efter arbejdet med de foreslåede aktiviteter. En sådan CT-nysgerrighed kan være en afgørende motivation for at beskæftige sig med Teknologiforståelsesfagets øvrige tre kompetenceområder og dermed for, at ele-

verne "opnår færdigheder og viden, således at de konstruktivt og kritisk kan deltage i udvikling af digitale artefakter og forstå deres betydning" (UVM, u.å.).

Refleksionsspørgsmål

- Prøv at lave en trinvis beskrivelse af en betaling med MobilePay. Hvornår er personen aktiv, hvornår er mobilen aktiv? Hvilke data indgår i en betaling? Prøv at lave en algoritme for mobilens aktiviteter.
- Udform en aktivitet, hvor eleverne går på jagt efter data som f.eks. skilte med tekst, plakater med billeder, indhold på en computerskærm. Hvilke typer af data finder eleverne (del f.eks. op i tekst, ikoner, tal, billeder)? Hvilke data er mon digitale, altså data som vises eller behandles af digital teknologi som computere? Tænk fx på grøntsagsvægten i et supermarked.

REFERENCER

Caprani, O. (2015). Mangfoldige læringsaktiviteter – ét robotbyggesæt. Læring & Medier (LOM) – nr. 14.

Caspersen, M. E., Iversen, O. S., Nielsen, M., Hjorth, A., & Musaeus, L. H. (2017). Computational thinking. Dolin, G. Holten Ingerslev & Hanne Sparholt Jørgensen (Red.), Gymnasiepædagogik. En grundbog. København: Hans Reitzels, 470-478.

Cuny, J., Snyder, L., & Wing, J. M. (2010). Demystifying computational thinking for non-computer scientists. Unpublished manuscript in progress, referenced in http://www. cs. cmu. edu/~ CompThink/resources/TheLinkWing. pdf.

Curzon, P., Bell, T., Waite, J., & Dorling, M. (2019). Computational Thinking. In The Cambridge Handbook of Computing Education Research. Cambridge University Press.

Denning, P. J., & Tedre, M. (2019). Computational Thinking. Mit Press.

Guzdial, M. (2019). A new definition of Computational Thinking: It's the Friction that we want to Minimize unless it's Generative. https://computinged.wordpress.com/2019/04/29/what-is-computational-thinking-its-the-friction-that-we-want-to-minimize/

Guzdial, M. (2008). Education Paving the way for computational thinking. Communications of the ACM, 51(8), 25-27.

Lee, I., Martin, F., Denner, J., Coulter, B., Allan, W., Erickson, J., ... & Werner, L. (2011). Computational thinking for youth in practice. Acm Inroads, 2(1), 32-37.

Musaeus, L. H., & Musaeus, P. (2019, February). Computational Thinking in the Danish High School: Learning Coding, Modeling, and Content Knowledge with NetLogo. In Proceedings of the 50th ACM Technical Symposium on Computer Science Education (pp. 913-919). ACM.

Papert, S. & Solomon, C. (1972). Twenty things to do with a computer. Educational Technology Magazine, s. 9-18.

Papert, S. (1980). Mindstorms: Children, computers, and powerful ideas. Basic Books, Inc.

Piaget, J. (1971). Psychology and epistemology: Towards a theory of knowledge (A. Rosin, Trans.). New York: Viking.

Resnick, M., & Rosenbaum, E. (2013). Designing for tinkerability. Design, make, play: Growing the next generation of STEM innovators, 163-181.

Sorteper (u.å.), http://spilregler.dk/sorteper/

UVM (u.å.), Teknologiforståelse læseplan https://emu.dk/sites/default/files/2019-02/GSK.%20L%C3%A6seplan.Tilg%C3%A6ngelig.%20Teknologiforst%C3%A5else.%20pdf.

Vendespil (u.å.), https://nuskalvilavebogstavsjov.wordpress.com/2011/02/14/vendespil-hjemmelavede-og-pa-mange-mader/

Wing, J. M. (2006). Computational thinking. Communications of the ACM, 49(3), 33-35.

Ole Caprani er lektor, Center for Computational Thinking & Design, Aarhus Universitet

Line Have Musaeus er ph.d.-studerende, Center for Computational Thinking & Design, Aarhus Universitet