Bacheloroppgave

NTNU
Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for datateknologi og informatikk

Deividas Marcenko
Kasper Djahangirloo Halvorsen

Anvendelse av programmering i språk på ungdom-/videregående skole nivå

Bacheloroppgave i Informasjonsbehandling
Veileder: Atle Nes
Mai 2023

NTNU
Kunnskap for en bedre verden

Deividas Marcenko
Kasper Djahangirloo Halvorsen

Anvendelse av programmering i språk på ungdom-/videregående skole nivå

Bacheloroppgave i Informasjonsbehandling
Veileder: Atle Nes
Mai 2023

Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for datateknologi og informatikk

NTNU

Kunnskap for en bedre verden

Sammendrag

Generelt sett har det i nyere tid blitt gjort endringer i ungdom- og videregående skolenes læreplaner for å øke fokus på dybdelæring i forskjellige fag. Et type fag som fremdeles ikke har fått dybdelæring integrert er språkfag.

Vi har derfor lagd et prosjekt hvor vi viser hvordan man kan integrere dybdelæring inn i disse fagene ved hjelp av programmering om det noen gang skulle bli utført.

Språkfag blir normalt ikke tenkt på i sammenheng med programmering siden det er et type fag med fokus på historie, litteratur og språk. Fag som normalt ikke involverer problemløsning gjennom algoritmer.

VI har derfor utforsket og funnet ut hvordan man kan integrere programmering innenfor slike fag ved å plukke ut og anvende oppgaver fra fagbøkene deres. Vi har lagd to oppgavehefter hvor den ene er for trinn 8-10 på Ungdomskolen og den andre er til Vg1-Vg3 på videregående skole. Heftet for ungdomskolen inneholder 12 oppgaver og videregående inneholder 11.

Heftene er tilpasset vanskelighetsgraden for trinnene de blir gitt til. Hver oppgave i heftene består av oppgavebeskrivelse, løsningsforslag, hele koden som blir brukt, og eksempel på hvordan output vil se ut.

Siden språkfags-lærere normalt ikke får noen opplæring i programmering, blir det referert til noen nettsider i prosjektet hvor man kan få både korte og lengre innføringer i det. Innføringene vil både vise hvordan programvarene funker og hvordan selve kodingen blir gjort.

I teoridelen blir det diskutert i mer detalj hva dybdelæring, algoritmisk tenkning og programmering betyr. Det blir også forklart hvordan oppgaver i språkfag som blir anvendt til programmering vil kunne gi elevene slik tenkning og læring.

Abstract

In recent times, changes have been made to the curricula of middle and high schools in order to increase focus on in-depth learning in various subjects. One type of subject that has not yet had in-depth learning integrated is language subjects.

Therefore, we have created a project to show how one can integrate in-depth learning into these subjects with the help of programming, if it should ever be carried out. Language subjects are not normally associated with programming since they are a type of subject focused on history, literature, and language. Subjects that do not normally involve problem-solving through algorithms.

We have therefore explored and found out how programming can be integrated into such subjects by selecting and applying tasks from their textbooks. We have created two task booklets, one for grades 8-10 in middle school and the other for grades 11-13 in high school. The booklet for middle school contains 12 tasks and the high school booklet contains 11. The booklets are adapted to the difficulty level of the grades they are given to. Each task in the booklets consists of a task description, a solution proposal, the entire code used, and an example of what the output will look like.

Since language teachers do not normally receive any training in programming, the project refers to some websites where both short and longer introductions to it can be obtained. The introductions will show both how the software works and how the coding itself is done. The theoretical part discusses in more detail what in-depth learning, algorithmic thinking, and programming mean. It also explains how tasks in language subjects that are used for programming can provide students with such thinking and learning.

Innholdsfortegnelse

Figurer

1 Introduksjon

1.1 Bakgrunn

Digitalisering av Norge sine skoler i de siste tiårene har økt betydelig mye med bruken av mobiltelefon og datamaskiner i undervisningen. En stor grunn til dette er regjeringens investering i digital infrastruktur (bærbare maskiner) på skolene (Universitetet i Oslo, 2022). Noe annet som drev digitalisering av skolene videre var covid-19 pandemien. Undervisning under pandemien måtte foregå på nett, noe som drev elever og lærere til å lære nye teknologiferdigheter. Dette åpnet muligheter for digitale undervisningsverktøy til å bli forbedret og brukt i større grad (Utdanningsdirektoratet, 2020).

Med tanke på økende digitalisering i fremtiden, burde elevene få utdanning innenfor teknologi for å få bedre digitale ferdigheter (Balanskat & Engelhardt, 2015). For å få bedre kunnskap av teknologien og den digitale infrastrukturen, må elevene også få undervisning i programmering sammen med de teoretiske deler av informatikk. «Programming is a key part of computer science and computing; it is a skill that cannot sit separately from the theoretical components of computing.» Ved å lære programmering, klarer eleven til å forstå digitale problemer på en større grad, og derfor løse disse problemene (Waite & Sentance, 2021).

I juni 2015 anbefalte Ludvigsen-utvalget på *Fremtidens skole* at det skulle vektlegges fire kompetanseområder i skolens faglige innhold (Sevik, 2016):

- fagspesifikk kompetanse
- kompetanse i å lære
- kompetanse i å kommunisere, samhandle og delta
- kompetanse i å utforske og skape

Når det kommer til fagspesifikk kompetanse, er det mulig å knytte programmering mot disse fire områdene og å videre knytte dem inn mot spesifikke fag. Et eksempel er matematikk, der elevene vil styrke matematiske kompetanse ved å knytte de teoretiske matematiske begreper mot en praktisk kontekst. Når det kommer til kompetanse i å lære, kan programmering hjelpe elevene til å planlegge hvordan de skal utføre koden - tenke algoritmisk. Når det kommer til samarbeid og kommunikasjon, kan programmering bli gjort sammen med andre elevene i form av samarbeid med koden eller spørre om hjelp. Når det

gjelder kompetanse i å utforske og skape, kan programmering være veldig åpenbar når det kommer til løsninger av problemet, det vil si elevene kan være kreative på måtene de bruker (Sevik, 2016).

1.2 Kunnskapsløftet

Flere undersøkelser viste at elevene fikk dårlig resultater i skolene, det var stor sosial ulikhet, og utdanningen ikke var tilpasset godt nok for alle. Derfor kom Stortinget ut med Stortingsmelding nr. 30 som presenterte ulike endringer innen skolene som: nye læreplaner, ny timefordeling, lokal varighet, og forsterkning av de fem grunnleggende ferdigheter (Kunnskapsdepartementet, 2004):

- Å kunne utrykke seg muntlig
- Kunne lese
- Kunne utrykke seg skriftlig
- Kunne regne
- Kunne bruke digitale verktøy

I skoleåret 2006-2007 ble kunnskapsløftet innført, der 1. -9. trinn i grunnskolen og Vg1 i videregående tar i bruk de nye læreplanene, mens 10. trinn og Vg2 skulle vente til 2007-2008, og Vg3 til 2008-2009 (Kunnskapsdepartementet, 2006, 2007).

Unntatt fra generell bruk av bærbare pc-er og hjelpemidlene som for eksempel digitale mapper og video/lyd/grafikk/tekst, var det ikke forklart hvordan bruk av digitale verktøy skulle tas i bruk i skolene. Spesielt i fagområder som for eksempel språkfag (Kunnskapsdepartementet, 2004).

1.3 Fagfornyelsen

Gjennom årene etter innføring av kunnskapsløftet, ble det merket at det fortsatt var dårlige resultater fra studenter når det kom til fullføring av videregående opplæring. Problemet med kunnskapsløftet var at de nye læreplanene hadde mange forskjellige upresise temaer/mål, og det var vanskelig for lærere å forstå hvordan de skulle lære disse temaene

til elever. Det var også vanskelig for alle elevene å lære like godt som hverandre. For å bekjempe dette kom Stortinget ut med en fornyelse av kunnskapsløftet- Meld.St.28 (Kunnskapsdepartementet, 2016).

Hovedmålet med fornyelsen var å sette søkelys på hvert enkelte fag, ved å forklare hva som skal læres i hvert av dem og legge til rette for dybdelæring. Det ble beskrevet hva som er viktigst i hvert fag slik at lærere kunne vite hva de skulle lære. I tillegg slik at elevene kunne ta inn kunnskapen uten å fokusere på for mange temaer. Fagene blir mer praktiske og knyttet sammen, og elevene kommer til å hjelpe å bestemme hva og hvordan de lærer. Dette hjelper med elevenes lyst til å lære. Det er også stort fokus på kildekritikk, det er viktig for elevene å kunne bestemme hva de kan stole på. Det nevnes også forsterkning av digitale ferdigheter og programmering innen fagfornyelsen, og «hvilke fag som har ansvar for opplæring i de ulike sidene/delene av ferdighetene» (Kunnskapsdepartementet, 2016; Utdanningsdirektoratet, 2021).

For å se på endringene og kjerneelementer innen språkfag, må vi se både på norsk, engelsk og fremmedspråk.

Kjerneelementer for norskfaget (Kunnskapsdepartementet, 2018):

- Tekst i kontekst
- Kritisk tilnærming til tekst
- Muntlig kommunikasjon
- Skriftlig tekstskaping
- Språket som system og mulighet
- Språklig mangfold

Endringer i norskfaget (Kunnskapsdepartementet, 2018):

- Mer eksperimentering
- Øve på akademisk skriving på videregående skolenivå
- Dypdykk i språk- eller litteraturhistorie, og sette disse inn i en språklig eller historisk sammenheng

Norskfaget er både et språkfag og et kulturfag, det vil si at elever vil lære både bruk av språket, men også litteratur historie og norsk-kulturen. Elever skal lære på utforskende

måte- ved å for eksempel drøfte, sammenligne, og reflektere. De skal også lære å være kildekritiske slik at de kan reflektere over troverdigheten til tekstene de leser (Utdanningsdirektoratet, 2019d).

Kjerneelementer for engelskfaget (Kunnskapsdepartementet, 2018):

- Kommunikasjon
- Språklæring
- Møte med engelskspråklige tekster

Endringer i engelskfaget (Kunnskapsdepartementet, 2018):

- Vektlegge engelsk som arbeidsspråk
- Vektlegge kommunikasjon mellom mennesker som ikke har engelsk som morsmål

I engelskfaget, ved siden av språklæring og lesing, skal elevene også lære på utforskende måter. De skal lære om samfunnet, levemåter, hvordan man kan bruke engelsk til å kommunisere og jobbe med tanke på kildekritikk. Elevene skal lære hvor nyttig engelsk språk ferdigheter er i samfunnet og i verden (Utdanningsdirektoratet, 2019b).

Kjerneelementer for fremmedspråk (Kunnskapsdepartementet, 2018):

- Kommunikasjon
- Interkulturell kompetanse
- Språklæring og flerspråklighet
- Språk og teknologi

Endringer i fremmedspråk (Kunnskapsdepartementet, 2018):

- Utvikle kunnskaper for å kommunisere hensiktsmessig muntlig og skriftlig
- Bruke språket både med og uten bruk av ulike medier og verktøy.

Når det kommer til fremmedspråk, er det viktigst for eleven å lære seg kommunikasjon. Eller forklart annerledes: «å forstå og å bli forstått». Det er også viktig å lære hvor nyttig flerspråklighet er som en ressurs i samfunnet (Utdanningsdirektoratet, 2019c).

Kjerneelementer for engelsk fordypning (Kunnskapsdepartementet, 2018):

- Kommunikasjon
- Språklæring
- Interkulturell kompetanse
- Språk og teknologi

Når det kommer til engelsk fordypning, blir det viktigst å arbeide praktisk. Det nevnes kreativ og kritisk bruk av teknologi og digitale ressurser til utforsking av språket. Elevene skal utvikle språkferdigheter ved å «trekke inn egne interesseområder fra både den virtuelle og virkelige verden» (Utdanningsdirektoratet, 2019a).

1.4 Læreplanene og utfordringer

Når det kommer til programmering i læreplaner for språkfag, kommer ordet programmering og algoritmisk tenkning nesten aldri opp. Når det gjelder digital kompetanse, blir digitale ressurser brukt kun til innhenting av informasjon i de fleste språkfagene. Set som fremdeles nevnes i de fleste språkfag-læreplaner er at elevene skal lære på utforskende måter, og det er der vi kan knytte språkfag mot programmering. De fleste språkfag kan regnes også som kulturfag, noe som utvider muligheten for at fagene kan knyttes til programmering (Utdanningsdirektoratet, 2019d).

Det som er viktig for programmering innen språkfag å lykkes er lærerkompetanse. For at lærere skal kunne undervise programmering, må de forstå det selv, og dette kan være vanskelig for lærere uten IKT kompetanse/kurs i programmering. Et annet problem som kan komme opp er at lærere kommer til å ta bort tiden fra språklæringen til å undervise i programmering. Hvis programmering ikke er tilpasset godt nok til pensum, kan det hende at viktige temaene i læreplanen kuttes ut for å gi plass til programmering (Balanskat & Engelhardt, 2015).

1.5 Oppgavens tittel

“Anvendelse av programmering i språk på ungdom-/videregående skole nivå” er arbeidstittelen som vi ble gitt i oppgaveteksten. Dette er den endelige tittelen vi går med siden den godt beskriver hva vi skal gjøre, nemlig anvende oppgaver i språkfag til programmeringsoppgaver for ungdom- og videregående skoler.

1.6 Problemstilling

Oppgaveteksten som vi ble gitt av oppgavestiller beskriver oppgaven slik:

“Hensikten med oppgaven er å finne ut hvilke oppgaver innen språk i skolen som egner seg best til programmering for å øke dybdeforståelse, engasjement og kreativitet hos elever, og anbefalte fremgangsmåter for implementasjon av disse."

Hensikten er som sagt at vi skal finne oppgaver i språkfagenes fagbøker/pensum som kan anvendes til programmering. Gjennom løsning av de anvendte oppgavene skal elevene kunne lære seg både språkfagets pensum og programmering samtidig. De vil dermed lære seg å bruke algoritmisk tenkning til å løse problemer/presentere arbeidet sitt.

Når programmering blir en del av fagets pensum (i dette tilfelle språkfag) trenger både fremtidige og eksisterende lærere å få grunnleggende opplæring i programmering. De må få en forståelse for koding og algoritmer slik at de kan forstå oppgavesettene selv. I tillegg må de være i stand til å hjelpe elever som selv kan ha vanskeligheter med programmering. Det blir derfor referert til noen nettsider i starten av kapittel 5 som kan gi lærere/elever en god innføring i de forskjellige programvarene som skal bli brukt.

Oppgaveheftene skal gi elevene muligheten til å løse/presentere oppgaver i språkfag gjennom programmering. Alle oppgavene i heftet vil ha en fremgangsmåte som viser hvordan man lager de nødvendige algoritmene og kommer fram til rett løsning med dem i tillegg til et fullt eksempel på koden og output.

2 Teori

Det som blant annet vil bli diskutert som en del av teorien-kapittelet er:

- Dybdelæringens viktighet i fagfornyelsen
- Hvordan utforskende læring blir brukt i skolen
- Hvordan problemløsning er en viktig metode for å hjelpe til med dybdelæring
- Hva er algoritmisk tenkning?
- Hvordan tenker man algoritmisk?
- Hva er programmering?
- Hvordan programmering skal bli brukt i språkfag

2.1 Dybdelæring

I Meld.St.28- «en fornyelse av kunnskapsløftet» ble det satt stort fokus på dybdelæring når det kommer til de nye læreplaner. Dybdelæring var grunnlaget for alle endringene som:

- mer praktiske fag
- redusert mengde av temaer i læreplanen
- bruke forskning som lærestrategi.

Det ble konkludert i fornyelsen at dybdelæring er viktig for elevenes fremtid, og for å komme frem til denne konklusjonen tok de inspirasjonen fra Ludvigsen-utvalget: NOU2014:7 «Elevenes læring i fremtidens skole - Et kunnskapsgrunnlag» og NOU 2015:8 «Fremtidens skole - Fornyelse av fag og kompetanser» (Kunnskapsdepartementet, 2016).

I følge NOU2014:7, betyr dybdelæring at «elevene utvikler forståelse av begreper og sammenhenger innenfor et fagområde». Det vil si at allerede kjente ideer og prinsipper kan knyttes til nye ideer, noe som hjelper med å løse ukjente problemer ved bruk av allerede forståtte prinsipper. «Dybdelæring innebærer at elevene bruker sin evne til å analysere, løse problemer og reflektere over egen læring til å konstruere helhetlig og varig forståelse». For at elevene skal lykkes med problemløsning i ukjente sammenhenger, er det viktig for dem å lære hvordan de kan lære i dybden (Ludvigsenutvalget, 2014).

Dybdelæring kan bli beskrevet i motsetning til overflatelæring som legger vekt på innlæring av fakta-kunnskap uten at eleven setter kunnskapen i en sammenheng. Uten dybdeforståelse, kommer elevene til å huske fakta og regler uten å forstå hva de betyr og hvordan disse kan bli brukt i forskjellige sammenheng (Ludvigsenutvalget, 2014).

Figur 1 Deep Learning Versus Traditional Classroom Practices

Learning Knowledge Deeply (Findings from Cognitive Science)	Traditional Classroom Practices (Instructionism)
Deep learning requires that learners relate new ideas and concepts to previous knowledge and experience.	Learners treat course material as unrelated to what they already know.
Deep learning requires that learners integrate their knowledge into interrelated conceptual systems.	Learners treat course material as disconnected bits of knowledge.
Deep learning requires that learners look for patterns and underlying principles.	Learners memorize facts and carry out procedures without understanding how or why
Deep learning requires that learners evaluate new ideas, and relate them to conclusions.	Learners have difficulty making sense of new ideas that are different from what they encountered in the textbook.
Deep learning requires that learners understand the process of dialogue through which knowledge is created, and they examine the logic of an argument critically.	Learners treat facts and procedures as static knowledge, handed down from an all-knowing authority.
Deep learning requires that learners reflect on their own understanding and their own process of learning.	Learners memorize without reflecting on the purpose or on their own learning strategies.

Kilde: (Sawyer, 2006)

I følge NOU2015:8, «Dybdelæring er like viktig for utvikling av kompetanse i alle fag, grunnskolefag så vel som fellesfag og programfag i videregående opplæring. Å lære og beherske fagenes metoder og tenkemåter er vesentlig for alle fagene på skolen». Et stort

problem som nevnes i utvalget er stofftrengsel i skolen, det er vanskelig å ta inn nytt lærestoff uten at noe annet tas ut. Dybdelæring og forståelse av temaer tar tid, så i stedet for å ta noe ut burde man prioritere hvilken metode eller fagstoff som skal oppnå ønskede kompetansemål (Ludvigsenutvalget, 2015).

Tettere sammenheng mellom fagene kan også bidra til mindre stofftrengsel. Hvis det er bestemt hvilke fag som har ansvar for hva, blir det mindre overlapping i fag, og derfor mer tid til å sette seg inn i fagene i dybden. Dybdeforståelse kan også skapes ved å sette fagkunnskaper i nye sammenhenger, og derfor er kreativitet også veldig viktig å tenke på når det kommer til læreplan (Ludvigsenutvalget, 2015).

2.1.1 Dybdelæring i språkfag

Når det kommer til språkfag, kan dybdelæringen bli styrket gjennom det disse fagene har til felles. Selv om språkfag som norsk, engelsk og fremmedspråk har kulturelle forskjeller, kan metoder elever bruker til å lære disse språk være ganske like. Med tettere samvirke mellom språkfag burde man også tenke på forskjellige læringsmetoder som elever kan bruke til å lære de forskjellige språk. Siden alle elevene lærer forskjellig, kan det være viktig å implementere nye måter å lære disse språkene på (Ludvigsenutvalget, 2015).

Med tanke på NOU2015:8 (Ludvigsenutvalget, 2015) og ideene om språkfagenes forsterkning av dybdelæring, ble oppgavene i dette prosjektet laget slik at det er en sammenheng mellom de forskjellige språkfagene. Det vil si: hvilket som helst språkfag kan ta i bruk disse oppgavene med små justeringer for å tilpasse aktuelle faget eller elevene.

2.2 Utforskende læring

Noe som ofte blir nevnt innen disse språkfagene og de nye læreplaner er utforskende læring. Utforskende læring handler om kritisk tenkning og problemløsning, og kreativitet og innovasjon (Ludvigsenutvalget, 2015).

Når det kommer til kritisk tenkning og problemløsning, er det viktig å kunne å analysere slik at eleven kommer opp med relevante spørsmål og strategier for å løse problemet. For å ta i bruk riktig informasjon er det viktig å være informasjon-/kilde-kritisk (Ludvigsenutvalget, 2015).

Når det gjelder kreativitet og innovasjon innen utforskende læring, er det viktig å være nysgjerrig og å komme opp med nye måter å løse problemet på. Det blir også viktig å ta initiativ til å sette i gang de nye løsningene (Ludvigsenutvalget, 2015).

2.2.1 Modellene innen utforskende læring

For å komme i gang med utforskende læring, kan man ta i bruk noen av de forskjellige læringsmodellene som (Universitetet i Oslo, 2023):

- 5E-modellen
- Sokratisk samtale
- Problembasert læring
- Entreprenørskap
- Nysgjerrigper
- Storyline
- Argument
- Prosjektarbeidsmetoden

Vi skal nå se på de forskjellige modellene og hvordan disse er relevante til utforskende læring og dette prosjektet.

5E modellen

5E modellen består av de fem fasene:

Fase 1- Engasjere (Engage): Ny tema eller spørsmål blir gitt til elevene slik at de blir interessert i å tenke videre på den (Universitetet i Oslo, 2021).

Fase 2- Undersøke (Explore): Elever skal utforske gjennom formulering av hypoteser, problemløsning og testing av ideene (Universitetet i Oslo, 2021).

Fase 3- Forklare (Explain): Eleven utvikler kunnskapen videre gjennom analyse og kritikk av data (Universitetet i Oslo, 2021).

Fase 4- Utvide (Expand): Eleven tar den nye informasjon og går i dybden på den ved å knytte informasjonen til nye situasjoner (Universitetet i Oslo, 2021).

Fase 5- Vurdere (Evaluate): Eleven og lærer går gjennom det eleven har lært (Universitetet i Oslo, 2021).

Selv om denne modellen først ble utviklet for undervisning i biologi, fant lærere ut at den kunne brukes i alle skolefag. For eksempel, når elevene skal lære seg om kulturen i språkfaget eller selve språket, kan de bruke 5E modellen. I fase 2 for eksempel, får de jobbet litt praktisk og undersøkende med temaet. Når det kommer til prosjektet, kan de bruke programmering i denne fasen for å jobbe og utforske litt praktisk. I fase 3- må de være kritiske mot data de får fra medelever og data de finner selv, noe som er nevnt som et kjernepunkt i de fleste språkfag (Universitetet i Oslo, 2021a).

Storyline

Når det kommer til språkfag, blir den mest brukte metoden storylinemetoden. Det er fire steg som er viktig for at metoden lykkes (Universitetet i Oslo, 2021b):

1. Visualisering og identifikasjon
2. Hendelser og nøkkelspørsmål
3. Fagsløyfer
4. Oppsummering og avslutning

Elevene skal etablere et univers der noe skjer og de må løse et problem eller svare på et spørsmål i det universet. For å svare på spørsmålet må eleven jobbe med faget og til slutt ende opp med å oppleve et eksisterende univers som de selv har diktet opp. Denne metoden kan øke dybdeforståelse i temaet ved å ta kunnskapen og sette den inn i det nye universet. Den kan også bidra til økt kreativitet siden eleven kommer opp med scenarioet selv (Universitetet i Oslo, 2021b).

Prosjektarbeidsmetoden

Denne metoden kan være nyttig hvis lærere har erfaring med elevaktive arbeidsformer. Denne metoden har fem prinsipper (Universitetet i Oslo, 2021):

1. Problemorientering
2. Produktorientering
3. Deltagerstyring eller fellesstyring
4. Tverrfaglighet og dermed faglighet og faglig kvalitet
5. Eksemplarisk innholdsvalg

Elevene skal få et problem eller en oppgave og skal lage et produkt som skal løse dette problemet. Elevene burde få hjelp fra lærere hvis det trengs. Prosjektet kan være tverrfaglig, noe som gir en mulighet til å ta i bruk andre knyttede fag og programmering i prosjektet. Elevene må også sette seg dypt inn i fagstoffet i stedet for å gå videre til nytt fagstoff (Universitetet i Oslo, 2021).

Problembasert læring

Man kan bruke problembasert læring i språkfag også: elevene kan komme til en kulturell forskjell, vanskelig grammatisk problem, eller et ord de ikke forstår. Problembasert læring kan kuttes inn i syv steg (Pettersen, 2017):

1. Oppfatning av situasjonen
2. Definer og avgrens hovedtemaer, problemer og eventuelle delproblemer
3. Mulige forklaringer, årsaker, sammenhenger og hypoteser
4. Diskusjon, elaborering og vurdering av egen kunnskap
5. Mål for videre læring: felles og individuelle læringsmål
6. Individuelle studier - ny kunnskap
7. Sammenfatning, integrering og anvendelse av ny kunnskap

Problembasertlæring kan knyttes også til algoritmisk tenkning, noe som blir snakket mer om i kapittel 2.3 Algoritmisk tenkning.

2.3 Algoritmisk tenkning

Futschek definerer en algoritme som: “en metode for å løse et problem som består av nøyaktige definerte funksjoner” (Futschek, 2006, s. 160).

Han går videre med å si at algoritmisk tenkning er en pool av evner som er koblet sammen til å konstruere og forstå algoritmer. Altså å ha evnen til å:

- Analysere gitte problemer
- Spesifisere problemet presist
- Finne grunnleggende handlinger som er tilstrekkelig for et gitt problem
- Konstruere en korrekt algoritme til et gitt problem ved å bruke grunnleggende handlinger
- Tenke på alle mulige spesielle og normale tilfeller av et problem
- Forbedre effektiviteten av en algoritme

(Futschek, 2006, s. 160)

Man kan kort si at algoritmisk tenkning er å kunne løse komplekse problemer ved å dele dem inn i mindre delproblemer som kan løses enklere. Tenkningen går også ut på å generalisere løsningen på problemene man har utført slik at man kan bruke det om igjen på liknende problemer. Dette er spesielt viktig å gjøre med delproblemer siden man ved å kombinere generaliserte løsninger av dem, kan løse større og mer komplekse problemer. (Utdanningsdirektoratet, 2019e)

I språkfag bruker man i utgangspunktet ikke algoritmisk tenkning siden pensumet mest går ut på å lære om kultur, historie og litteratur for det gitte språket. Det å lære og reflektere over historien og kulturen til et land gjennom tidene involverer i utgangspunktet ikke problemløsning ved hjelp av algoritmer. Det går mer ut på å bruke historisk og kronologisk tenkning. Altså at man memoriserer og reflekterer over hvilke hendelser som ledet til den neste og hvorfor det skjedde i utgangspunktet**.** (American Historical Association, 1884)

Det å lære seg å skrive skjønnlitteratur handler om å bruke kreativ tenkning og forståelse av språk og grammatikk. I programmering bruker man selvsagt kreativitet til å for eksempel designe utseende på en nettside. Forskjellen er at man må skrive alt ved hjelp av koding, mens man i språk bare skriver ord og setninger enten digitalt eller fysisk på papir. Man trenger dermed ikke å bruke algoritmisk tenkning i språkfag siden man løser problemene sine ved å bruke et tastatur/blyant til å skrive setninger. Hvor man i programmering må finne ut hvordan man skal kode en serie instruksjoner slik at ideene blir presentert korrekt. Når man skriver digitalt blir ekstra tekniske aspekter som skriftstørrelse og marg mulig å kontrollere uten koding siden programvarer som Word har lagt dem inn som elementer som kan bli tilpasset etter brukerens behov.

I programmering er algoritmisk tenkning derimot helt essensielt. Når man programmerer har man alltid som mål å konstruere noe og finne ut hvordan man kan oppnå det. Dette kan være alt fra å lage en database av produkter for en nettbutikk, til å posisjonere et bilde-element korrekt på en nettside. Det er her man bruker algoritmisk tenkning til å løse komplekse og små problemer.

En stor del av oppgavene som er laget i dette prosjektet krever bruk av algoritmisk tenkning fra elevene. Blant annet vil de trenge å generalisere algoritmer/koder slik at de kan gjenta dem for å løse et større problem bestående av flere delproblemer.

2.4 Programmering

Rossen definerer programmering som: “utforming av dataprogram som avgjør hvordan elektronisk apparatur skal fungere mens programmet er aktivt eller kjører.” (Rossen, 2022)

Forklart i mer dybde, blir programmering gjort ved å sette opp en serie med instruksjoner som kontrollerer en maskin og hvordan den skal reagere på inndata og input fra brukeren. Måten å sette instruksjonene opp på avhenger av hvilken type datamaskin man programmerer på, og det er forskjellige programmeringsspråk man kan kode program på i dag. (Rossen, 2022)

Eksempler på slike språk er lavnivåspråk som “C” hvor instruksjonene som blir skrevet, ikke ser veldig forskjellig ut fra den endelige maskinkoden som kommer fra den. Språket er derfor vanskelig for vanlige mennesker å forstå. Med høynivåspråk som “C++” derimot, skrives programmet med koding som mye enklere kan bli forstått av de fleste mennesker. Det er derfor vanskeligere og mer tidkrevende med lavnivåspråk, selv om man får mye dypere kontroll over maskinvare. Koding som blir gjort i enkelt tilgjengelige programvarer som Visual Studio Code er da skrevet i høynivåspråk. Instruksjonene som blir skrevet i de forskjellige språkene blir alltid lagret i filer som enten hver for seg eller sammen med flere filer utgjør et spesifikt system eller program. (Rossen, 2022) (Nätt, 2023)

Selv om de fleste kun tenker på koding når de hører ordet programmering, er det viktig å få frem at programmering også er en prosess som foregår utenfor maskinvaren. Før man faktisk skriver instruksjonene må man først løse problemet i sitt eget hode ved å analysere og dele den opp i mindre delproblemer. Med veldig kompliserte problemer er det også smart å skissere løsningen ut nøye før man begynner å kode. Det er denne måten å løse problemer på som kalles algoritmisk tenkning. Når man løser de små delproblemene og koder instruksjonene inn, vil man ende opp med å løse det store problemet som en helhet. Gjenbruk av koder og løsninger er essensielt i programmering og en annen sentral del av algoritmisk tenkning som blir kalt generalisering. (Utdanningsdirektoratet, 2019e)

Programmering blir som sagt brukt som en metode for å løse problemer. Siden det ikke har blitt brukt innenfor pensumet i noen språkfag ennå, blir det plukket ut oppgaver fra fagbøker i språkfag som blir anvendt til programmering. Oppgaver som består av skjønnlitterære og historiske problemer, skal derfor løses ved hjelp av programmering.

3 Valg av Metode

Design Science Research

For valg av metode, ble det valgt Design Science Research metode. Design Science Research, eller DSR, er et problemløsningsparadigme som prøver å forbedre teknologi og vitenskapelig kunnskap gjennom å lage artefakter som er ment til å løse spesifikke problemer (Brocke et al., 2020).

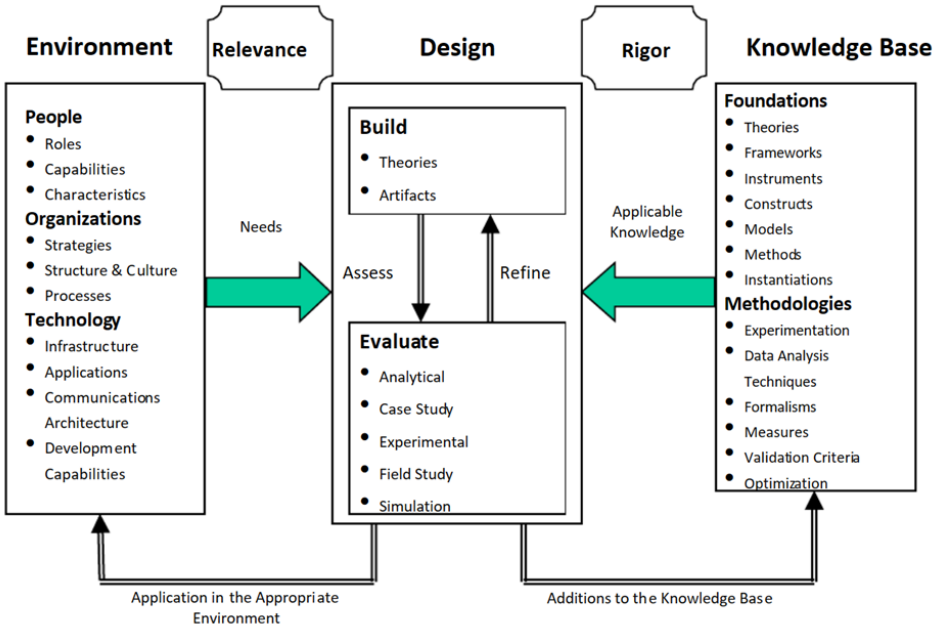
DSR har et rammeverk som består av Environment, Design, og Knowledge Base (Brocke et al., 2020).

Innen Environment, finnes alle parter som har relevansen til problemet. Folk, organisasjoner og teknologier med tanke på ønsker av selve løsningen, hjelper til å bestemme hva som er kravet for selve artefaktet. Når det kommer til selve prosjektet, ble kravene til artefaktet skrevet ned i visjonsdokumentet (Brocke et al., 2020).

Innen Knowledge Base, finner man all relevant informasjon som hjelper til å bygge selve artefaktet. For selve prosjektet, var det relevant å finne eventuelle mulige programmeringsoppgaver innen språkfag, noe som ikke var lett tilgjengelig i språkfagsundervisning/læreplaner på tiden av rapportens skriving. Som Knowledge Base, bruktes det relevant fagstoff og læreplaner innen språkfag til å lage nye programmeringsoppgaver (Brocke et al., 2020).

Design består av selve artefaktet, som i dette prosjektet er oppgaveheftet av oppgavene som lærere kan bruke til å lære seg programmering, og bruke disse oppgavene innen undervisningen av elever (Brocke et al., 2020).

Figur 2 Design Science Research Framework

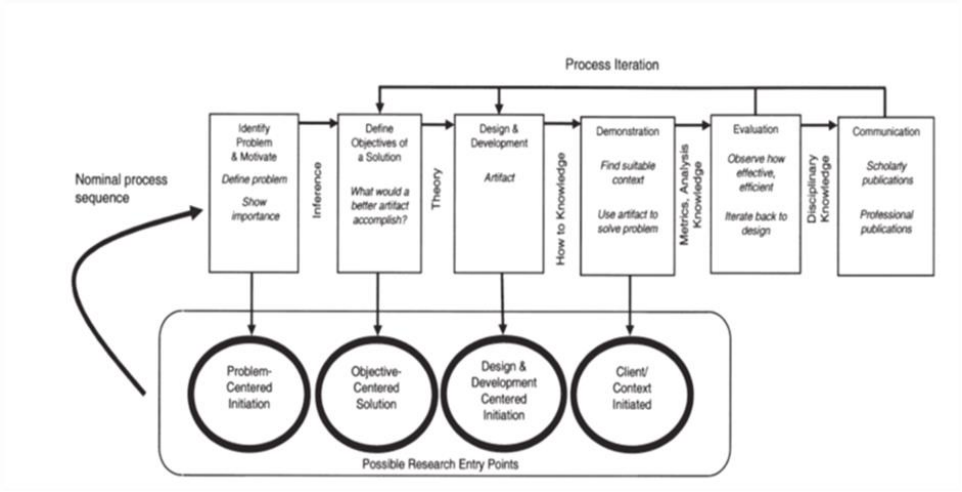


Source: (Brocke et al., 2020)

Gjennomføring av DSR består av seks prosesser:

- Problem identification and motivation
- Define the objectives for a solution
- Design and development
- Demonstration
- Evaluation
- Communication

Figur 3 DSR Methodology Process Model



Source: (Brocke et al., 2020)

I dette prosjektet blir bare de tre første prosesser gjennomført;

1. Problemet blir identifisert, det vil si mangel på programmering og algoritmisk tenking innen språkfag blir forklart.

2. Artefaktet blir definert i visjonsdokumentet, hvilke krav som er sett, og hvordan det skal gjennomføres.

3. Selve artefaktet, oppgaveheftet, er laget. Artefaktet blir laget samt med dens ønskede funksjonalitet og struktur.

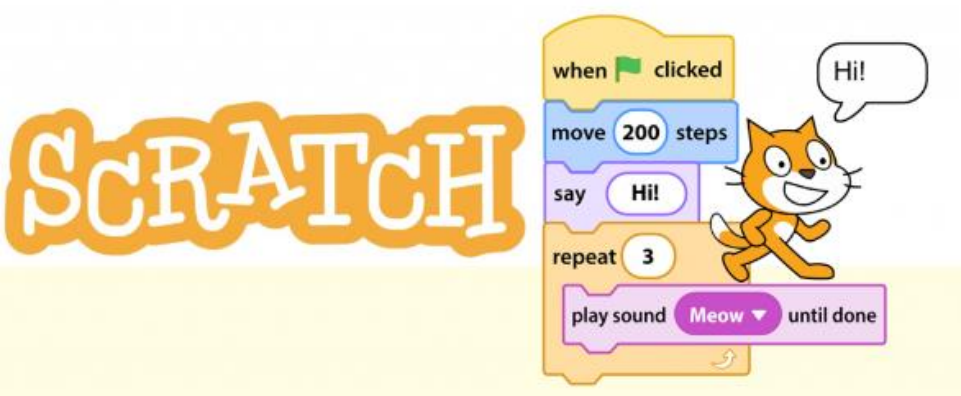
Prosesser fire til seks blir utført etter at prosjektet er gjennomført på grunn av stor arbeidsmengde og begrenset tid.

4 Valg av teknologi

4.1 Scratch

Scratch ble valgt som programvare for Ungdomskolen siden den fungerer som en enklere versjon av en normal programvare for koding av Javascript som Visual Studio Code. I stedet for tekstbasert programmering, går den ut på å bruke blokkbasert programmering hvor blokkene er fargekodet og inneholder ferdig definerte funksjoner. Scratch er derfor en mer visuell form for programmering og vil derfor passe godt som en introduksjon til koding for yngre elever. Ungdomsskoleelevene vil dermed kunne visualisere koden før de fortsetter over til mer tung og komplisert tekstbasert koding i programvarer som VSCode.

Figur 4 Scratch-blokker



Kilde: (TechnoKids, 2018)

I tillegg til at den er enkel å bruke, fungerer det teatralske aspektet av den godt med de skjønnlitterære oppgavene fra Ungdomskolens pensum og fagbøker. Nettsiden til Scratch og dens verktøy er også lett tilgjengelig og gratis å bruke for alle, og man kan lage seg en bruker for å lagre alle prosjektene/oppgavene sine. Det blir derfor ikke like komplisert og tidskrevende for elever/lærere å måtte laste ned programvare og navigere gjennom lokale filer på skolens datamaskiner.

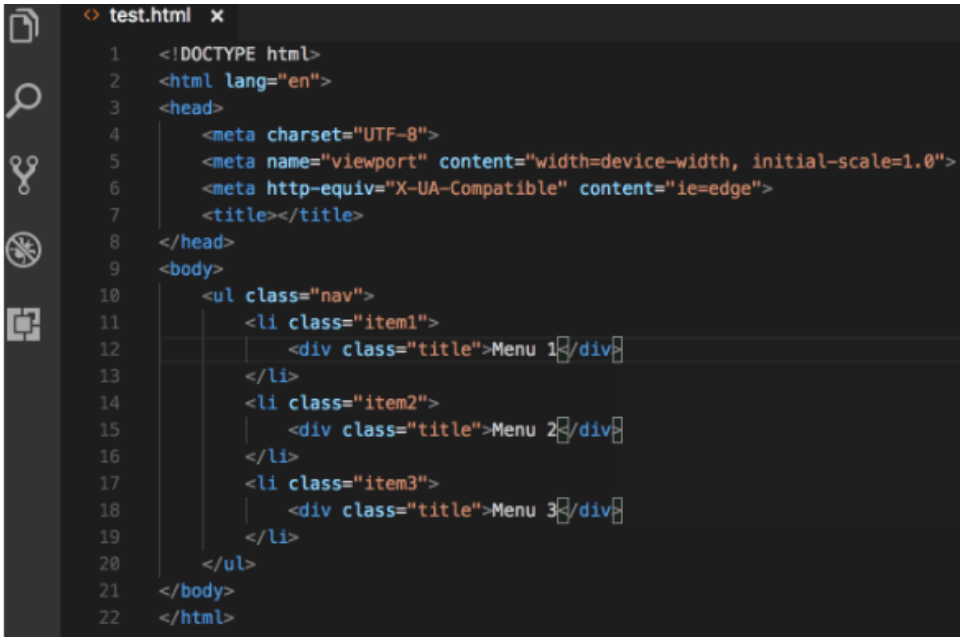
4.2 Visual Studio Code

Vi valgte å bruke Visual Studio Code som programvare for Videregående- og ungdomskoler siden det er et av de vanligste programvarene som blir brukt for standard koding av

Javascript i dag. VS Code vil også kunne brukes til mer komplisert koding enn Scratch som vil dukke opp i videregående skolens oppgavehefte. Samtidig skal VS Code brukes til litt enklere programmering i ungdomskolens oppgavehefte siden visse algoritmer ikke er mulig å realisere i Scratch. Brukergrensesnittet til VS Code er oversiktlig med hvordan de forskjellige elementene er fargekodet og designet slik at elevene vil få god oversikt over alt de skriver.

Eventuelt kan man bruke en online versjon av en lignende programvare kalt JS Bin. Her kan man programmere innenfor Javascript på samme måte som VS Code, men på en åpen nettside i stedet for lokalt på egen/skolens datamaskin.

Figur 5 VS Code HTML eksempel



Kilde: (Visual Studio Code, 2015)

5 Resultater

Selv om det finnes mange forskjellige språkfag både i ungdoms- og i videregående skole, er fremgangsmåte til å lære disse språkene ganske lik. For eksempel - eleven skal lære grammatikk på en veldig lik måte både på spansk og på engelsk selv om de grammatiske reglene er forskjellige. Eleven skal lære om språkenes litteraturhistorie på lignende måte både på norsk og engelsk. Eleven lærer også om kulturen både på norsk og fransk. Det ble derfor bestemt å ikke lage forskjellige oppgavehefter til hvert språk, men i stedet å ta de

vanligste språkene - norsk og engelsk, og lage oppgavene basert på disse. De fleste oppgavene i oppgaveheftet kan lett bli gjort om til lignende oppgaver i de andre språkfagene.

Oppgaveheftet ble delt inn i to deler: en for ungdomsskole, og en for videregående skole. Den ene forskjellen mellom disse to er at videregående skole oppgaver er vanskeligere, gjennom ungdomsskole oppgavene kommer elevene til å få kjennskap til programmering slik at de kan takle de vanskeligere oppgavene i videregående skole senere. Den andre forskjellen mellom disse to kapitlene er at ungdomsskole del, inneholder både Scratch og Javascript oppgavene, mens videregående skole del kun inneholder Javascript. Det ble bestemt at det er viktigere for ungdomsskole elever å kunne visualisere koden for å forstå den enklere, noe som kan bli gjort med Scratch. Javascript oppgavene blir brukt til ungdomsskole elever også slik at de kan få forståelse og kjennskap til Javascript før de takler vanskeligere videregående skole oppgaver.

For å finne oppgavene, ble det først forsket på om det allerede finnes løsninger i form av språkfag kode oppgaver. Etter at ingen relevante oppgaver kom opp, ble det bestemt å bruke relevante fagbøker i norsk og engelsk, og lage kodeoppgavene ut fra de eksisterende oppgavene i disse bøkene.

Bøkene som ble brukt er:

- Stages 8. Engelsk for ungdomstrinnet av Synnøve Pettersen og Felicia Røkaas.
- Stages 9. Engelsk for ungdomstrinnet av Synnøve Pettersen og Felicia Røkaas.
- Access to English av Richard Burgess og Theresa Bowles Sørhus.
- Edge - programfaget engelsk 1, vg2/3 - studieforberedende utdanningsprogram av Ian Underwood, Arne Birkeland, Svein Arild Pettersen, og Siri Hunstadbråten.
- Kontekst 8-10 – for Ungdomskolen av Kathinka Blichfeldt, Tor Gunnar Heggem og Åslaug Huseby
- Panorama Norsk Vg 1 | Studieforberedende av Marianne Røskeland, Jannike Ohrem Bakke, Liv Marit Aksnes og Gunnstein Akselberg
- Panorama Norsk Vg 2 | Studieforberedende av Marianne Røskeland, Jannike Ohrem Bakke, Liv Marit Aksnes, Gunnstein Akselberg og Sveinung Time

- Panorama Norsk Vg 3 | Studieforberedende av Marianne Røskeland, Jannike Ohrem Bakke, Liv Marit Aksnes, Gunnstein Akselberg og Sveinung Time

Hver oppgave inneholder beskrivelse av oppgaven, forklaring av løsningen, fremgangsmåte gjennom algoritme, programkode, og eksempel output.

Ungdomsskole delen inneholder 9 Javascript oppgavene, og 3 Scratch oppgavene.

Videregåendeskole delen inneholder 11 Javascript oppgavene.

Det er forventet at lesere allerede er kjent med Scratch og Javascript. I tilfelle leseren er nybegynner, anbefales det å lære Javascript fra sider som JavaScript Tutorial (w3schools.com) og Javascript basics (developer.mozilla.org)

For å få en kort innføring i hvordan Scratch sin nettside funker, kan man gå til Scratch sin forside Scratch.mit.edu, klikke på "Programmering" øverst på venstre hjørnet av siden, gå til "Veiledninger" på toppen av siden og klikke på videoen "Getting Started – ASL". For en lengre og mer utfyllende innføring kan man bruke oppgaver.kidsakoder.no til å få opp en side som forklarer om nettsidens funksjoner i mer detalj.

5.1 Ungdomsskole Scratch del

5.1.1 Dramatiser debatt

Denne oppgaven kan endres for å passe andre språkfag også.

Oppgave:

Lag et program som dramatiserer en debatt der en debattant prøver å være så saklig som mulig og holder seg til nivå 4-5 i pyramiden, mens den andre er usaklig og er på nivå 1-3.

Oppgaven er hentet fra (Blichfeldt, Heggen og Huseby, 2020, s. 225)

Løsning:

Start med å skrive et manus til debatten som oppgaven beskriver, slik at du er klar for å implementere det inn i programmet.

For å lage et program som dramatiserer en samtale på Scratch, må man først opprette to eller flere figurer avhengig av hvor mange karakterer som skal snakke. En bakgrunn kan også bli lagt til om man har lyst til å gi en spesifikk setting til debatten. Man må dermed lage et program for begge figurene som får dem til å si en rekke replikker etter hverandre. I tillegg må man kode programmet slik at hver figur viser replikken fram et visst antall sekunder samtidig som den andre figuren venter på å gi sin replikk til den andre er ferdig.

Om man har lyst til at karakterene skal være plassert på en viss posisjon i starten av programmet, er det bare å plassere en "gå til" blokk i starten av koden under start-blokken. Der kan du skrive inn posisjonen du vil at figuren skal være i på x- og y-aksen til scenen.

Man kan selvsagt velge å eksperimentere med funksjoner og blokker ved å for eksempel ha figurene bevege seg rundt omkring i rammen. I så fall kan man bruke blokken "gå x sekunder til x: x-akse posisjon og y: y-akse posisjon".

Algoritme:

1. Opprett eventuell bakgrunn som ønskes til dramatiseringen

2. Opprett 2 valgfrie figurer i et prosjekt på Scratch, enten ved å bruke eksisterende figurer på nettsiden eller ved å laste opp egne valgte bilder.

    2.1 Start instruksjonene til begge figurene med en *"når flagg klikkes"* start-blokk og en *"gå til x"* blokk fra *"Sansing"* kategorien hvor du skriver inn hvor du vil figurene skal bli plassert i starten av dramatiseringen

    2.2 Start med å plassere en *"si i x sekunder"* blokk fra *"Utseende"* kategorien under *"gå til x"* blokken til Figur 1.

        2.2.1 Skriv figurens første replikk i *"si" boblen* og antall sekunder du har lyst til at replikken skal vises fram i boblen mellom *"i"* og *"sekunder"*

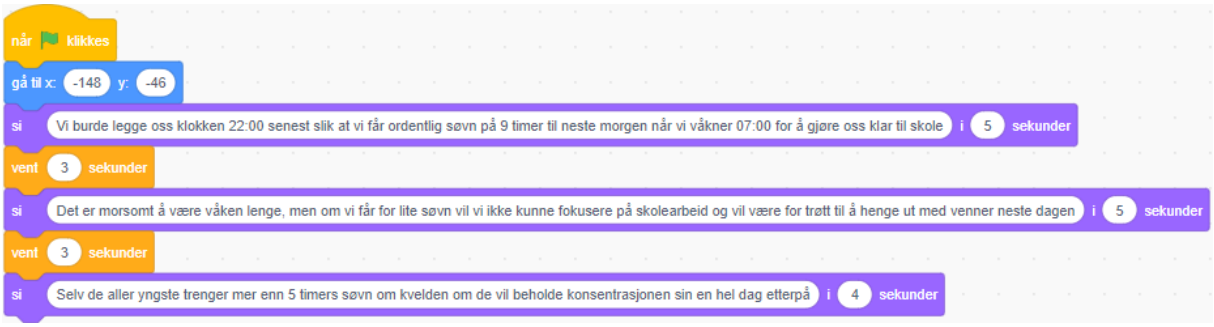
2.3 Gå til Figur 2 og plasser en *"vent x sekunder"* blokk under *"gå til x"* blokken.

2.3.1 Skriv deretter inn det samme antallet sekunder som ble skrevet inn i *"si i x sekunder"* blokken fra Figur 1 .

2.6 Om man kjører programmet nå, vil Figur 1 si replikken sin samtidig som Figur 2 venter like mange sekunder på å gjøre noe som Figur 1 sin snakkeboble vises i rammen.

2.7 For å fortsette dialogen plasserer man en *"si i x sekunder"* under *"vent x sekunder"* blokken til Figur 2. Repeter samme prosess som fra 2.2-2.5 og veksle mellom hver figur. Gjør dette helt til man er ferdig med dialogen man ønsker for oppgaven.

Figur 1 sin kode:



Figur 6

Figur 2 sin kode:

Figur 7

Eksempel output:

Figur 8

5.1.2 Quiz

Denne oppgaven kan endres for å passe andre språkfag også.

Oppgave:

Lag et program som fungerer som en quiz hvor brukeren blir spurt en serie med spørsmål som må svares korrekt på en etter en for å vinne. Quizen kan bli lagd for ethvert gitt tema som blir undervist i språkfaget i den gitte undervisningsperioden. For eksempel kan quizen være om historie, litteraturhistorie eller språk til språkfaget det blir undervist i. Etter å ha lagd programmet skal den bli gitt til en medelev som skal prøve å løse quizen.

Oppgaven er ikke hentet fra en spesifikk side i noen av fagbøkene, men gjelder som en quiz for historie/litteraturhistorie/språk som tema i alle språkfag.

Løsning:

Når man lager et quiz-program, trenger man ikke å ha med ekstra teatralske aspekter som figurer og bakgrunner utenom en hovedfigur som kan stille brukeren spørsmålene. For å lage et quiz-program, må man gi en figur en "spør x og vent" blokk som gjør til at brukeren kan skrive inn et svar som figuren kan ta imot. Deretter bruker man "hvis/ellers" funksjon/blokk til å si hva som skal skje om svaret er korrekt eller galt.

Man skal gi quizen til en medelev som skal løse den. Før man gir quizen til medeleven går man i fullskjerm på visningen av scenen slik at medeleven ikke skal kunne se koden og svarene. Om medeleven svarer rett, vil programmet gå til å stille det neste spørsmålet og om det blir feil må brukeren starte programmet på nytt. Medeleven skal da ta quizen helt til han/hun svarer rett.

Algoritme:

1. Ha en figur klar og start koden med en *"start" blokk*.

2. Plasser en *"spør x og vent"* blokk fra *"sansing"* kategorien under den, hvor du skriver inn det første spørsmålet til quizen du har skrevet.

3. Plasser en *"hvis/ellers"* blokk under *"spør x og vent"* blokken

    3.1 Ta en *"x=x"* blokk fra "*Operatør"* kategorien og plasser den inn i det åpne rommet ved siden av der det står *"hvis"* i starten av *"hvis/ellers"* blokken

        3.1.1 Plasser *"svar"* blokken fra *"Sansing"* kategorien inn i starten av operatøren og skriv inn svaret på spørsmålet på slutten av operatøren.

    3.2 Plasser en *"si x i x sekunder"* blokk imellom *"hvis"* og *"ellers"* delen av *"hvis/ellers"* blokken

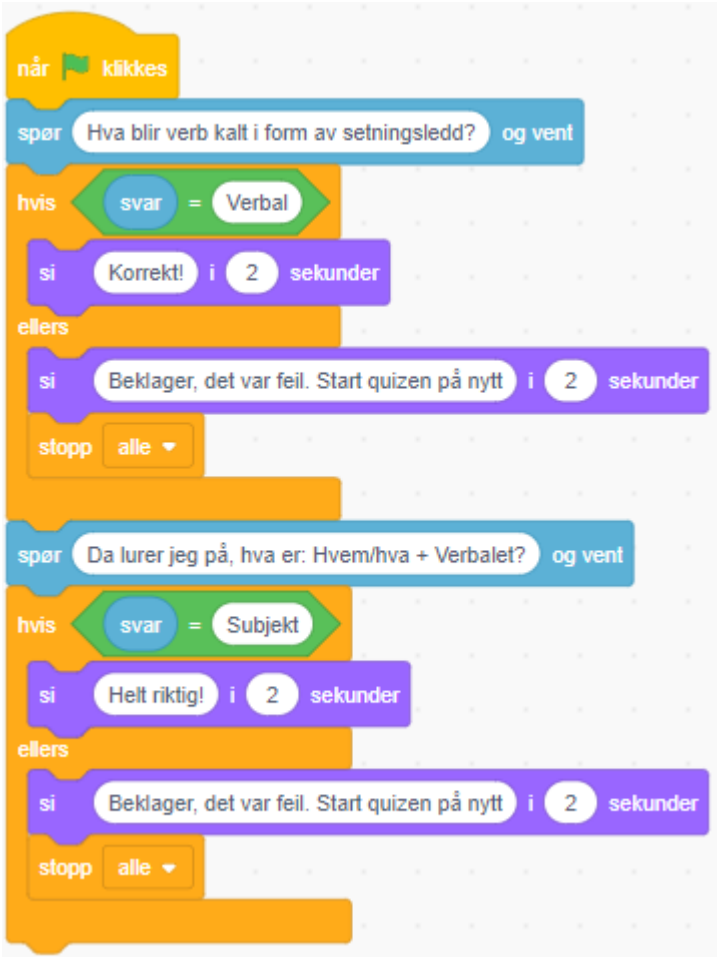
3.2.1 Skriv et ord/en setning som bekrefter at brukeren har svart rett på spørsmålet inn i *"si-boksen"* og skriv antall sekunder du vil at replikken skal vises frem i *"sekund-boksen"*

3.3 Plasser en *"si x i x sekunder"* blokk imellom *"ellers"* delen og slutten av *"hvis/ellers"* blokken

3.2.1 Skriv et ord/setning som bekrefter at brukeren har svart feil på spørsmålet inn i *"si-boksen"* og skriv antall sekunder du vil at replikken skal vises frem i *"sekund-boksen"*

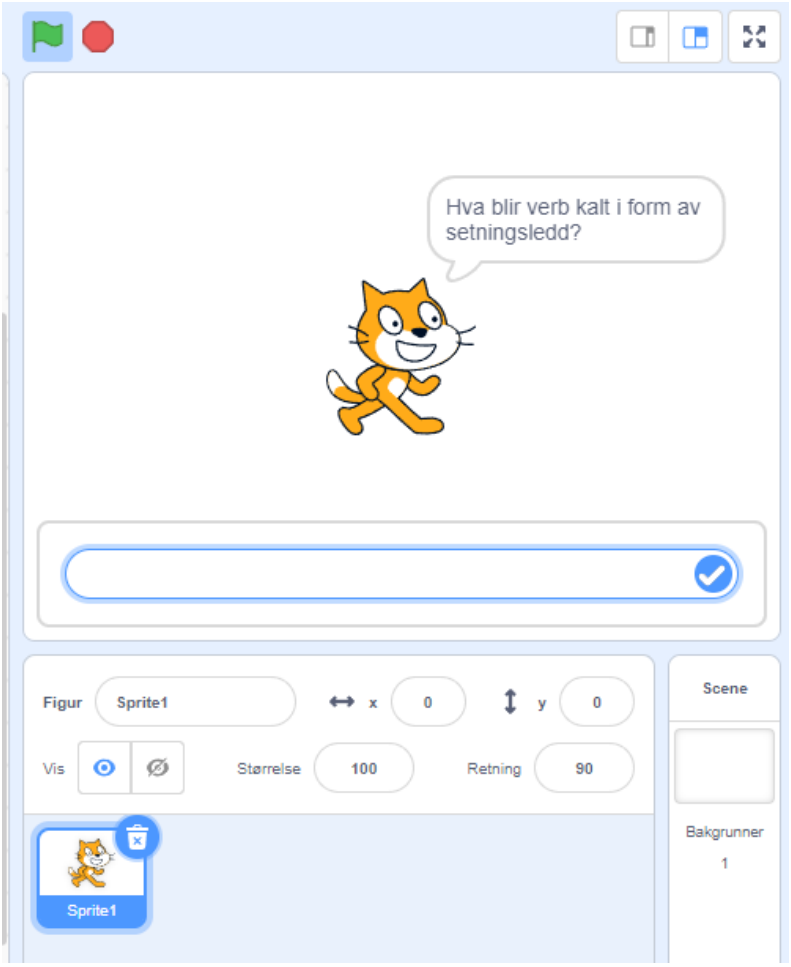
3.4 Plasser en *"stopp x"* blokk under den forrige *"si x i x sek"* blokken og sett alternativet til *"alle"*

3.5 Gjenta prosessen fra 3.1-3.4 for hvert spørsmål du skal still i quizen og plasser dem etter hverandre i koden.



Figur 9

Eksempel output:



Figur 10

5.1.3 Dramatisering av historisk figurs liv

Denne oppgaven kan endres for å passe andre språkfag også.

Oppgave:

Lag et program som kort dramatiserer høydepunktene til en litteraturhistorisk figur sitt liv. Dette gjelder for figurer i det gitte språkfaget

Oppgaven er hentet fra (Blichfeldt, Heggen og Huseby, 2020, s. 46)

Løsning:

Måten man velger å fremstille høydepunktene til en historisk figur på er relativt fritt. Som et eksempel kan det likevel vises hvordan man dramatiserer livet til den historiske personen ved å ha personen selv fortelle og gå gjennom det for publikumet.

For å vise høydepunktene i personen sitt liv kan man godt være kreativ ved å laste opp et bilde av den ekte personen som skal presenteres og gjøre han/hun til en figur. Deretter kan man ha figuren fortelle brukeren om de store hendelsene og milepælene i livet sitt gjennom replikker. Samtidig kan man endre bakgrunnen for hver milepæl det blir snakk om slik at den passer til hva det blir fortalt om.

Algoritme:

1. Last opp et bilde av den historiske figuren som skal fortelle om livet sitt.

2. Sørg for at du har de nødvendige bakgrunnene for hvert høydepunkt i livet hans/hennes enten lastet opp eller at de allerede er en del av Scratch sitt lager.

3. Start koden til figuren med en *"start"* blokk og en *"gå til x"* blokk hvor du skriver posisjonen du ønsker at figuren skal stå i starten av dramatiseringen.

3.1 Etterpå plasserer man en *"bytt til bakgrunn x"* blokk fra kategorien *"Utseende"*

3.1.1 Her kan man klikke på og velge hvilken bakgrunn man vil skal vises i løpet av koden som kommer etter denne blokken.

3.1.2 For å få bakgrunnene sine vist frem som en del av alternativene må man laste opp egne bakgrunnsbilder eller velge bakgrunner som allerede er på Scratch.

3.1.3 Man laster dem opp/velger dem ved å klikke på knappen i nederste høyre hjørnet av nettsiden som heter *"Velg et bakgrunnsbilde"*

3.2 Etter å ha valgt bakgrunn for høydepunktet man skal dramatisere, tar man en *"Si x i x sekunder"* blokk og plasserer den under *"bytt til bakgrunn x"* blokken.

3.2.1 Skriv ned hva den historiske figuren skal si om livet sitt i *"si"* delen av blokken og antall sekunder den skal vises som en snakkeboble i *"sekunder"* delen av blokken.

3.2.2 Det er lurt å kode inn flere *"si"* blokker etter hverandre med litt og litt replikker i stedet for en lang en slik at det blir mer engasjerende å se på dramatiseringen. Avhengig av hvor mye replikker man vil ha med er det bare å plassere flere *"si"* blokker etter hverandre

3.3 Når man er ferdig med et høydepunkt kan det være fint med litt variasjon i den visuelle presentasjonen. Man kan variere litt ved å ha figuren bevege seg til en annen posisjon i rammen ved bruk av *"gli x sekunder til x: x-posisjon og y: y-posisjon"* blokken fra *"Sansing»* kategorien.

3.3.1 Med denne blokken får du figuren til å flytte seg på en normal bevegelig måte fra foreløpig posisjon til gitt posisjon i løpet av x antall sekunder. Bevegelsen kan være fint å ha for å som sagt få visuelt engasjement fra publikum.

3.4 Etter å ha flyttet figurens posisjon kan man gå gjennom den samme prosessen som tidligere fra 3.1-3.3. Man fortsetter å bytte til nye bakgrunner og skriver flere replikker som skal vise de neste høydepunktene i figurens liv.

4. For å ha enda mer visuell variasjon mellom hvert høydepunkt kan man også inkludere andre figurer/kunstverk/objekter fra personens høydepunkt inne i presentasjonen.

4.1 For å gjøre dette kan man laste opp/velge flere figurer og skrive egen kode for når og hvor de skal dukke opp, i tillegg til om de skal ha noen replikker.

4.1.1 For at figur nr 2 (som for eksempel kan være et av figurens kunstverk) skal dukke opp i et visst høydepunkt, kan man legge in blokken *"når bakgrunn bytter til x".*

4.1.2 Velg hvilken bakgrunn man vil at figur 2 sine funksjoner skal starte på

4.1.3 Plasser dermed en *"Gå til x- og y-posisjon"* blokk under forrige blokk og skriv posisjonen du vil den skal dukke opp i.

4.1.4 Ta en *"vis"* blokk fra kategorien *"Utseende"* og plasser den under forrige blokk slik at figuren nå blir synlig i rammen.

4.1.5 Avhengig av om man vil at figuren skal stå stille eller ha en replikk, velger man enten *"si x i x sekunder"* eller *"vent x sekunder"* blokken/blokker. Uansett hva man velger, må man huske på å sørge for at handlingene til figur 1 og 2 ikke krasjer.

4.1.5.1 Om figur 2 skal stå stille og forsvinne når høydepunktet er ferdig, må man plassere en *"vent"* blokk etter *"vis"* blokken og sette antall sekunder til antall sekunder det tar for Figur 1 å bli ferdig med alle replikkene sine om høydepunktet.

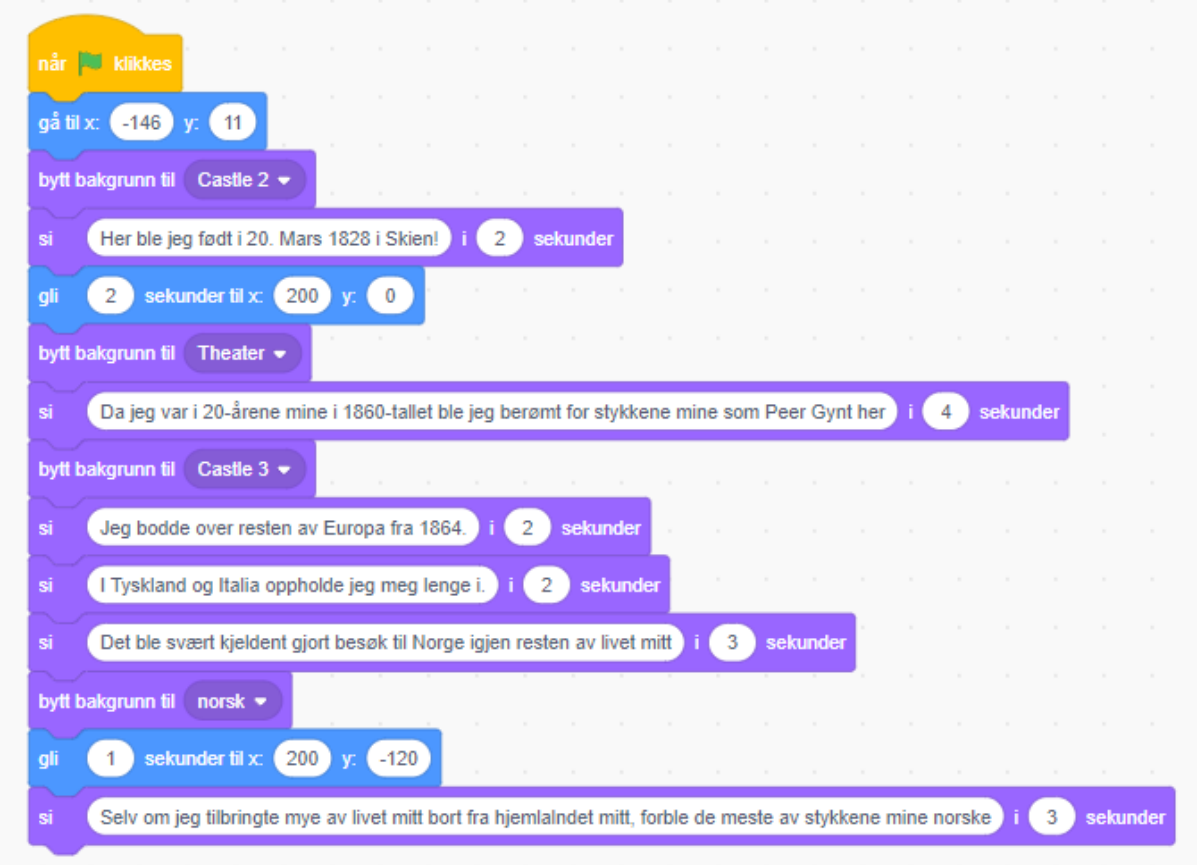
4.1.5.1.1 Deretter plasserer man en *"skjul"* blokk fra *"Utseende"* kategorien under *"vent"* blokken slik at figuren forsvinner fra rammen når den er ferdig brukt.

4.1.5.2 Om Figur 2 skal ha dialog med Figur 1 eller ha egne replikker for seg selv, må man sørge for å plassere en *"vent"* blokk i koden til Figur 1 slik at Figur 2 kan si replikkene sine uten avbrytelser fra Figur 1. Dette gjøres med begge to en etter en om de skal ha dialog med hverandre

4.1.5.2.1 Når bruken av Figur 2 er over, bruker man igjen en *"skjul"* blokk for å fjerne den fra rammen

5. Framgangsmåten på oppgaven er som sagt ganske fri og man eksperimentere med at flere figurer beveger seg og har dialog med hverandre i hvert høydepunkt. De mest grunnleggende funksjonene å følge er i uansett tilfelle i punkt 1-4.

Figur 1 sin kode:

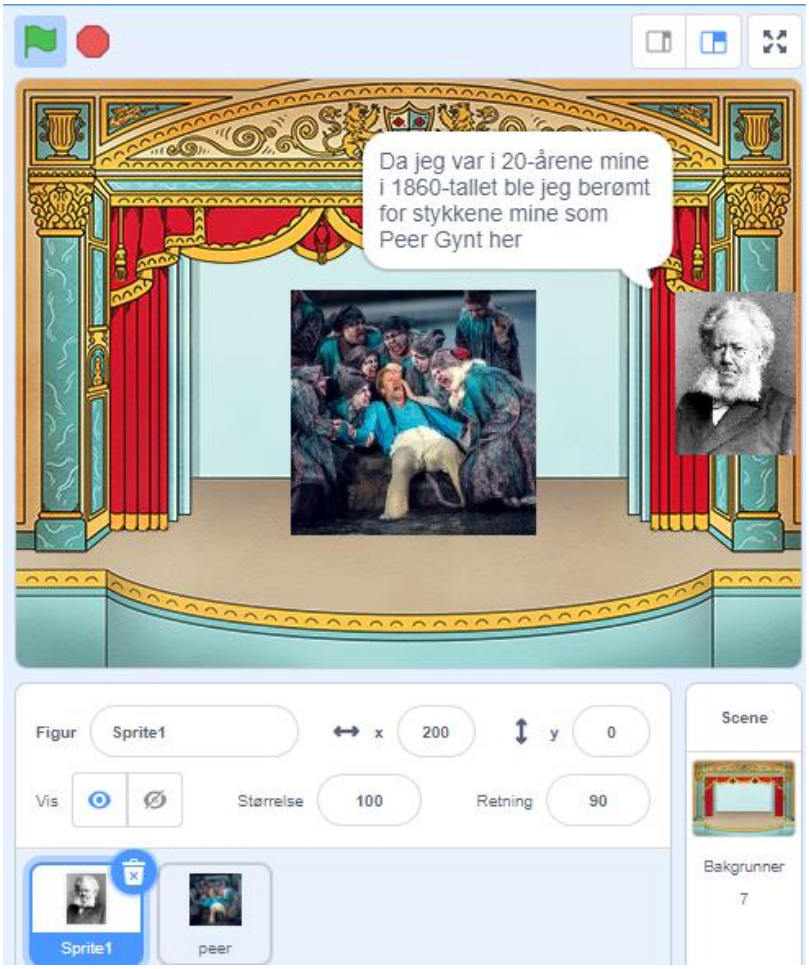


Figur 11

Figur 2 sin kode:

Figur 12

Eksempel output:



Figur 13

5.2 Ungdomsskole Javascript del

5.2.1 ex-

I engelskspråket, når du legger til en «ex-» foran et ord, betyr det at de ikke lenger er «dette».

For eksempel: Ordet «poser» blir om til «ex-poser»

Oppgave:

Lag et program som tar et ord og legger til en «ex-» foran ordet.

Oppgaven er hentet fra (S. Pettersen & Røkaas, 2020)

Løsning:

For å lage et program som kan ta inn et ord og endre den, må vi lage en input boks der du kan skrive hvilket som helst ord inn i. Vi må også lage kode for hvordan det kan legges en «ex-» foran ordet. Man kan for eksempel lage en funksjon som tar inputen fra input boks, legger til en «ex-» foran, og sender det til en annen tekst boks. Vi må også lage kode for å kjøre selve funksjonen, det kan gjøres for eksempel med hjelp av en knapp.

Algoritme:

1. Lag et HTML5 dokument.

2. Lag en *<input type='text'>* for å skrive inn ord.

    2.1. Lag en *id='word'* i input slik at ordet kan hentes ut til funksjonen.

3. Lag en *<p>* tekst boks.

    3.1 Lag en *id='output'* innen denne boksen slik at det nye ordet som blir laget innen funksjonen kan vises.

4. Lag en funksjon *addEx().*

    4.1 definer ordet *'newWord'* og hent inn ordet fra inputen ved hjelp av *document.getElementById('word').value*.

    4.2 legg til *'ex-'*til *'newWord'*

    4.3 sende det nye ordet til *<p>* ved hjelp av

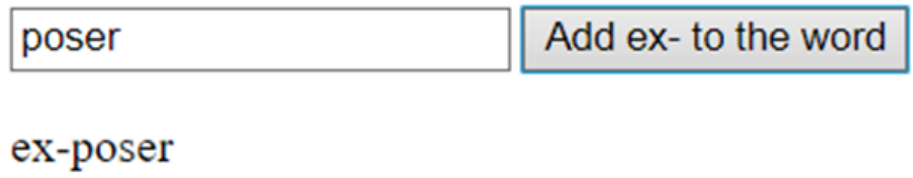
        document.getElementById('output').innerHTML.

5. Lag en knapp som kan kalle funksjonen *<button onclick='addEx()'>*

```
<!DOCTYPE html>
<html lang="en">
<body>
<!-- Brukeren kan skrive inn ordet her -->
<input type="text" id="word">
```

```
<!-- Brukeren kan kjøre funksjonen med knappen -->
<button onclick="addEx()">Add ex- to the word</button>
<!-- Output vises her -->
<p id="output"></p>

<script>
//Funksjonen for å legge til en ex- til et ord
function addEx(){
    //Tar input og legger den inn i newWord
    let newWord = document.getElementById("word").value;
    //Legger ex- til newWord og sender det til output <p>
    document.getElementById("output").innerHTML= "ex-"+newWord;
}
</script>
</body>
</html>
```

Eksempel output:



Figur 14

5.2.2 Gbp til nok

Når det kommer til engelskspråklige land, blir det viktig å vite forskjellen mellom valutaer og hvordan man kan regne disse hvis man skal reise eller forstå utenlandske nyheter.

Denne oppgaven kan bli brukt i andre språkfag også.

Oppgave:

Lag et program som tar gbp (british pound), og regner det om til nok (norske krone).

Oppgaven er hentet fra (S. Pettersen & Røkaas, 2020)

Løsning:

For å lage et program som kan ta inn et nummer og regne ut en annen, må vi først ha en input boks som kan ta inn et nummer. Etter det må vi lage en output boks som skal vise nok. Vi må også lage en funksjon som skal endre gbp til nok. Etter det kan vi lagge en knapp som skal kjøre selve funksjonen.

Algoritme:

1. Lag et HTML5 dokument.

2. Lag en *<input type='text'>* for å skrive inn et nummer.

    2.1. Lag en *id='gbp'* i input slik at nummeret kan hentes ut til funksjonen.

3. Lag en *<p>* tekst boks.

    3.1 Lag en *id='output'* innen denne boksen slik at det nye nummeret som blir laget innen funksjonen kan vises.

4. Lag en funksjon *exchangeRate().*

    4.1 definer ordet *'num'* og hent inn nummeret fra inputen ved hjelp av *document.getElementById('gbp').value*.

    4.2 definer ordet 'total' og multipliser den med valutakursen.

    4.2 gjør om desimal tall av *'total'* til 2 ved hjelp av *.toFixed(2)* og legg til *'nok'* på slutten for å vise at nummeret er norske krone

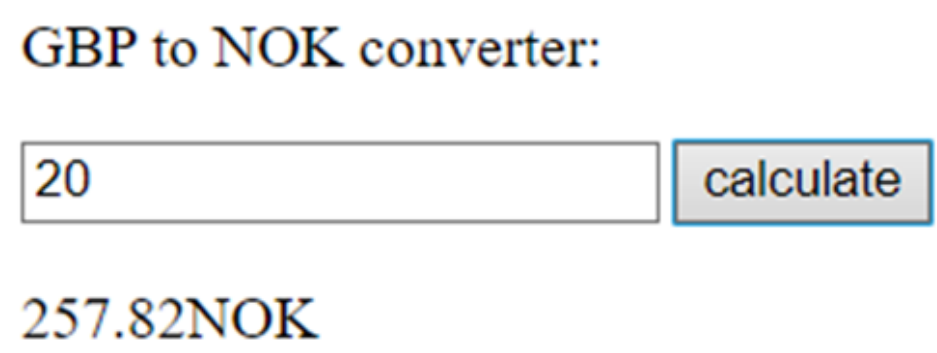
    4.3 sende det nye nummeret til *<p>* ved hjelp av *document.getElementById('output').innerHTML*.

5. Lag en knapp som kan kalle funksjonen *<button onclick=' exchangeRate()'>*

```
<!DOCTYPE html>
<html lang="en">
<body>
<p>GBP to NOK converter:</p>
```

```
<!-- Brukeren kan skrive inn gbp her -->
<input type="text" id="gbp">
<!-- Brukeren kan kjøre funksjonen med knappen -->
<button onclick="exchangeRate()">calculate</button>
<!-- Output vises her -->
<p id="output"></p>

<script>
//Funksjonen for å endre gbp til nok
function exchangeRate(){
    //Tar input og legger den inn i num
    let num = document.getElementById("gbp").value;
    //ganger num med valutakursen
    let total= num *12.8909;
    //Gjør om desimal tall til 2 og sender det nye nummeret til output <p>
    document.getElementById("output").innerHTML = total.toFixed(2)+"NOK";
}

    </script>
</body>
</html>
```

Eksempel output:



Figur 15

5.2.3 -es and -ies

I engelskspråket:

Når vi skal legge -s til verb som slutter på vislelyd (s, z, x, sh, ch) eller o, får verbet -es på slutten (S. Pettersen & Røkaas, 2020).

Når vi skal legge til -s til verb som slutter på konsonant + y, blir y til i, slik at endingen blir -ies (S. Pettersen & Røkaas, 2020).

Denne oppgaven kan endres for å passe andre språkfag også.

Oppgave:

Lag et program som tar et verb, og legger til den korrekte endingen -es eller -ies.

Oppgaven er hentet fra (S. Pettersen & Røkaas, 2020)

Løsning:

For å lage et program som kan ta inn et ord og endre den, må vi lage en input boks der man kan skrive hvilket som helst verb inn i. Vi må også lage kode for hvordan det bestemmes om det skal legges -es eller -ies til ordet. Det kan gjøres med en if- funksjonen. Vi må også lage koden for hvordan selve ordet blir endret. Etter det kan vi sende svar til en output tekst boks. Vi må også lage kode for å kjøre selve funksjonen, det kan gjøres for eksempel med hjelp av en knapp.

Algoritme:

1. Lag et HTML5 dokument.

2. Lag en *<input type='text'>* for å skrive inn verbet.

    2.1. Lag en *id='verb'* i input slik at ordet kan hentes ut til funksjonen.

3. Lag en *<p>* tekst boks.

    3.1 Lag en *id='output'* innen denne boksen slik at det nye verbet som blir laget innen funksjonen kan vises.

4. Lag en funksjon *find().*

4.1 definer ordet *'output'* for senere

4.2 definer ordet *'input'* og hent inn verbet fra inputen ved hjelp av *document.getElementById('verb').value*.

4.3 lag en if- setning.

4.3.1 først skal vi bestemme om verbet skal ende med -ies. *Input.endsWith('y')*

4.3.1.1 ta *output* og legg *input* inn i den. Ta vekk den siste bokstaven i ordet ved hjelp av *.slice(0,-1)* og legg til *+'ies'*

4.3.2 For å bestemme om ordet skal ende med -es kan vi bare skrive *else* som skal endre resten av ord som ikke ender med *y,* til -es

4.3.2.1 ta *output* og legg *input* inn i den. Legg til *+'es'* til *input*

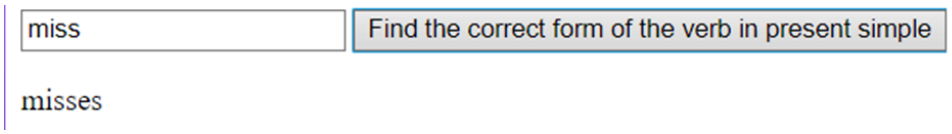
4.4 sende det nye verbet til *<p>* ved hjelp av *document.getElementById('output').innerHTML*.

5. Lag en knapp som kan kalle funksjonen *<button onclick=' find()'>*

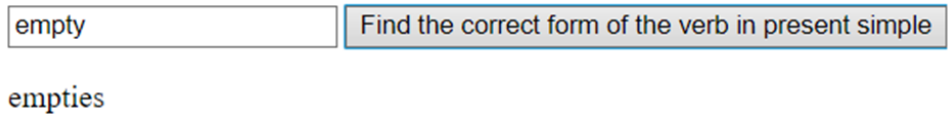
```
<!DOCTYPE html>
<html lang="en">
<body>
<!-- Brukeren kan skrive inn verbet her -->
<input type="text" id="verb">
<!-- Brukeren kan kjøre funksjonen med knappen -->
<button onclick="find()">Find the correct form of the verb in present
simple</button>
<!-- Output vises her -->
<p id="output"></p>

<script>
//funksjonen for å endre verbet
function find(){
    //definerer output
    let output;
    //definerer input og legger verbet inn i den
```

```
    let input= document.getElementById("verb").value;
    //bestemmer om endingen skal være -ies
    if (input.endsWith("y")){
        //tar vekk den siste bokstaven fra ordet og legger til "ies"
        output= input.slice(0,-1)+"ies";
    //bestemmer om endingen skal være -es
    } else {
        //legger til "es"
        output= input+"es"
    }
    //sender det nye verbet til output <p>
    document.getElementById("output").innerHTML = output;
}
</script>
</body>
</html>
```

Eksempel output:



Figur 16



Figur 17

5.2.4 Plural nouns

I engelskspråket:

Flertallsform blir vanligvis laget ved at vi legger til -s (S. Pettersen & Røkaas, 2020).

Hvis ordet ender med vislelyd, legger vi til -es (S. Pettersen & Røkaas, 2020).

Hvis ordet ender med en konsonant + y, erstatter vi y med i og legger til -es (S. Pettersen & Røkaas, 2020).

Hvis ordet ender med -f eller -fe, endrer vi f til v og legger til -es (S. Pettersen & Røkaas, 2020).

Denne oppgaven kan endres for å passe andre språkfag også.

Oppgave:

Lag et program som bestemmer riktig form av flertall.

Oppgaven er hentet fra (S. Pettersen & Røkaas, 2020)

Løsning:

For å lage et program som kan ta inn et ord og endre den, må vi lage en input boks der du kan skrive hvilket som helst ord inn i. Vi må også lage kode for hvordan det bestemmes hvilken ending som skal legges til slutten av ordet. Dette kan gjøres med en if- funksjonen. Vi må også lage koden for hvordan selve ordet blir endret. Etter det kan vi sende svar til en output tekst boks. Dermed må det lages kode for å kjøre selve funksjonen, det kan for eksempel gjøres ved hjelp av en knapp.

Algoritme:

1. Lag et HTML5 dokument.

2. Lag en *<input type='text'>* for å skrive inn ordet.

    2.1. Lag en *id='noun'* i input slik at ordet kan hentes ut til funksjonen.

3. Lag en *<p>* tekst boks.

    3.1 Lag en *id='output'* inne i denne boksen slik at det nye substantivet som blir laget inne i funksjonen kan vises.

4. Lag en funksjon *find().*

4.1 definer ordet *'output'* for senere

4.2 definer ordet *'input'* og hent inn verbet fra inputen ved hjelp av *document.getElementById('noun').value*.

4.3 lag en if- setning.

4.3.1 først skal vi bestemme om ordet skal ende med -es. *.endsWith("ss")||input.endsWith("sh")||input.endsWith("ch")||input.endsWith("x")||input.endsWith("z")*

4.3.1.1 ta *output* og legg *input* inn i den. Legg til *+'es'* til *input*

4.3.2 Så skal vi bestemme om verbet skal ende med -ies. *Input.endsWith('y')*

4.3.2.1 ta *output* og legg *input* inn i den. Ta vekk den siste bokstaven i ordet ved hjelp av *.slice(0,-1)* og legg til *+'ies'*

4.3.3 Så skal vi bestemme om verbet skal ende med -ves. *.endsWith("f")* og *(input.endsWith("fe")*

4.3.3.1 ta *output* og legg *input* inn i den. For ordet som ender med -f: ta vekk den siste bokstaven i ordet ved hjelp av *.slice(0,-1)* og legg til *+'ves'.* For ordet som ender med -fe: ta vekk de to siste bokstavene i ordet ved hjelp av *.slice(0,-2)* og legg til *+'ves'*

4.3.4 For å bestemme om ordet skal ende med -s kan vi bare skrive *else* som skal endre resten av ord til -s endingen.

4.3.4.1 ta *output* og legg *input* inn i den. Legg til *+'s'* til *input*

4.4 send det nye substantivet til *<p>* ved hjelp av *document.getElementById('output').innerHTML*.

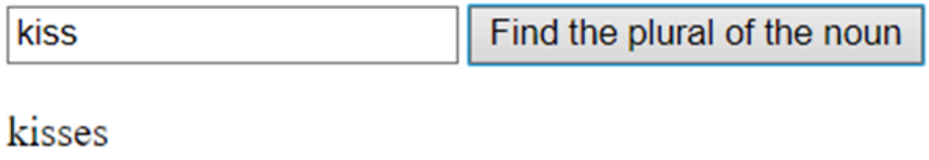
5. Lag en knapp som kan kalle funksjonen *<button onclick=' find()'>*

```
<!DOCTYPE html>
<html lang="en">
<body>
<!-- Brukeren kan skrive inn substantivet her -->
<input type="text" id="noun">
```

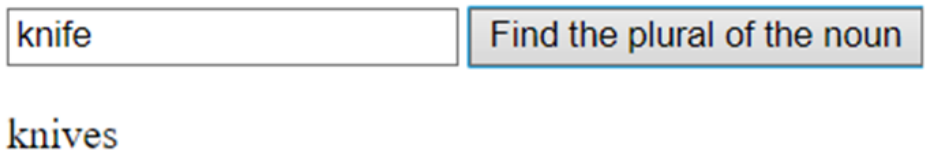
```
<!-- Brukeren kan kjøre funksjonen med knappen -->
<button onclick="find()">Find the plural of the noun</button>
<!-- Output vises her -->
<p id="output"></p>

    <script>
        //funksjonen for å endre verbet
        function find(){
        //definerer output
        let output;
        //definerer input og legger substantivet inn i den
        let input= document.getElementById("noun").value;
        //bestemmer om endingen skal være -es
        if
(input.endsWith("ss")||input.endsWith("sh")||input.endsWith("ch")||input.endsW
ith("x")||input.endsWith("z")){
        //legger til "es"
        output= input+"es"
        //bestemmer om endingen skal være -ies
        } else if (input.endsWith("y")){
        //tar vekk den siste bokstaven fra ordet og legger til "ies"
        output= input.slice(0,-1)+"ies";
        //bestemmer om endingen skal være -ves
        } else if(input.endsWith("f")){
        //tar vekk den siste bokstaven fra ordet og legger til "ves"
        output= input.slice(0,-1)+"ves";
        //bestemmer om endingen skal være -ves
        } else if (input.endsWith("fe")){
        //tar vekk de to siste bokstavene fra ordet og legger til "ves"
        output= input.slice(0,-2)+"ves";
        //bestemmer om endingen skal være -s
        }else {
        //legger til "s"
        output= input+"s"
        }
        //sender det nye substantivet til output <p>
        document.getElementById("output").innerHTML = output;
        }
```

```
    </script>
</body>
</html>
```

Eksempel output:



Figur 18



Figur 19

5.2.5 Sentence to question

I engelskspråket:

Det kan endres posisjonen av *is* eller *are* i setningen for å gjøre det om til et spørsmål

Oppgave:

Lag et program som gjør setningen om til et spørsmål ved bruk av *is* eller *are.*

Oppgaven er hentet fra (S. Pettersen & Røkaas, 2020)

Løsning:

For å lage et program som kan ta inn en setning og endre den, må vi lage en input boks der du kan skrive inn en setning. Vi må også lage kode for hvordan det bestemmes om det

finnes en *is* eller *are* i setningen. Dette kan gjøres med en if- funksjonen. Vi må også lage koden for hvordan selve setningen blir endret. Etter det kan vi sende svar til en output tekst boks. Vi må også lage kode for å kjøre selve funksjonen, det kan for eksempel gjøres med hjelp av en knapp.

Algoritme:

1. Lag et HTML5 dokument.

2. Lag en *<textarea>* for å skrive inn lengre setninger.

    2.1. Lag en *id='sentence'* i input slik at ordet kan hentes ut til funksjonen.

3. Lag en *<p>* tekst boks.

    3.1 Lag en *id='output'* innen denne boksen slik at den nye setningen som blir laget innen funksjonen kan vises.

4. Lag en funksjon *toQuestion().*

    4.1 definer ordet *'output'* for senere

    4.2 definer ordet *'input'* og hent inn verbet fra inputen ved hjelp av *document.getElementById('sentence').value*.

    4.3 lag en if- setning.

        4.3.1 først skal vi bestemme om setningen har ordet *is*. *.includes(" is ")*

            4.3.1.1 ta *output* og legg *input* inn i den. Ta ut ordet *is* fra inputen ved hjelp av *.replace(' is ', ' ').* Og legg til *Is* og *?* til setningen.

        4.3.2 så skal vi bestemme om setningen har ordet *are*. *.includes(" are ")*

            4.3.2.1 ta output og legg input inn i den. Ta ut ordet *are* fra inputen ved hjelp av .replace(' are ', ' '). Og legg til *Are* og *?* til setningen.

        4.3.3 Til slutten skal vi ved hjelp av *else* bestemme hva som skal gjøres hvis *is* eller *are* ikke finnes.

4.3.3.1 ta *output* og skriv hva som skal skrives ut hvis *is* eller *are* ikke finnes

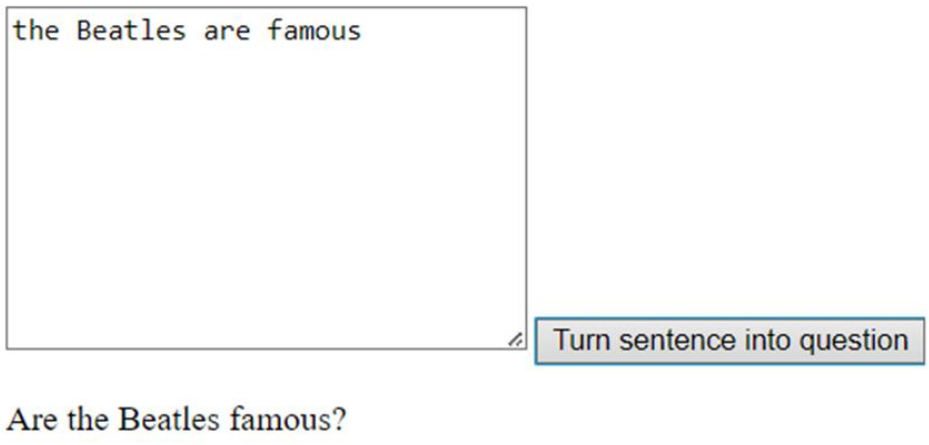
4.4 send den nye setningen til *<p>* ved hjelp av *document.getElementById('output').innerHTML*.

5. Lag en knapp som kan kalle funksjonen *<button onclick=' toQuestion()'>*

```
<!DOCTYPE html>
<html lang="en">
<body>
    <!-- Brukeren kan skrive inn setningen her -->
    <textarea name="sentence" id="sentence" cols="30" rows="10"></textarea>
    <!-- Brukeren kan kjøre funksjonen med knappen -->
    <button onclick="toQuestion()">Turn sentence into question</button>
    <!-- Output vises her -->
    <p id="output"></p>

    <script>
        //funksjonen for å endre setningen
        function toQuestion(){
            //definerer output
            let output;
            //definerer input og legger setningen inn i den
            let input = document.getElementById("sentence").value;
            //finner ut om det finnes ordet "is" inn i setningen
            if (input.includes(" is ")){
                //Tar ut "is" fra setningen og legger den foran setningen.
                output="Is "+ input.replace(" is "," ")+"?";
            //finner ut om det finnes ordet "are" inn i setningen
            } else if (input.includes(" are ")){
                //Tar ut "are" fra setningen og legger den foran setningen.
                output="Are "+ input.replace(" are "," ")+ "?";
            //bestemmer hva som skal gjøres hvis "is" eller "are" finnes ikke
            } else {
                output="Cannot find is or are"
            }
            //sender den nye setningen til output <p>
```

```
            document.getElementById("output").innerHTML = output
        }
    </script>
</body>
</html>
```

Eksempel output:



Figur 20

5.2.6 Comparing adjectives

Denne oppgaven kan endres for å passe andre språkfag også.

Oppgave:

Lag et program som skriver komparativ og superlativ form av adjektiver som ender med -y.

Oppgaven er hentet fra (S. Pettersen & Røkaas, 2020)

Løsning:

For å lage et program som kan ta inn et ord og skrive komparativ form av den, må vi først lage en input boks der du kan skrive ordet inn i. Vi må også lage kode for hvordan det bestemmes om ordet har en -y endingen. Dette kan gjøres med en if- funksjonen. Vi må

også lage koden for hvordan adjektivenes forskjellige former blir laget. Etter det kan vi sende svar til en output tekst boks. Vi må også lage kode for å kjøre selve funksjonen, det kan for eksempel gjøres med hjelp av en knapp.

Algoritme:

1. Lag et HTML5 dokument.

2. Lag en *<input type='text'>* for å skrive inn ordet.

    2.1. Lag en *id='adjective'* i input slik at ordet kan hentes ut til funksjonen.

3. Lag en *<p>* tekst boks.

    3.1 Lag en *id='output'* innen denne boksen slik at de nye form av adjektivet som blir laget innen funksjonen kan vises.

4. Lag en funksjon *compare().*

    4.1 definer ordet *'output'* for senere

    4.2 definer ordet *'input'* og hent inn adjektivet fra inputen ved hjelp av *document.getElementById('adjective').value*.

    4.3 lag en if- setning.

        4.3.1 først skal vi bestemme om adjectivet ender med -y. *.endsWith('y')*

            4.3.1.1 ta *output* og legg *input* inn i den tre ganger. La første inputen være sånn den er, for på den andre skal vi ta vekk den siste bokstaven og legge til en *'ier',* og på den siste skal vi ta vekk den siste bokstaven og legge til en *'iest'*

        4.3.2 Til slutten skal vi ved hjelp av *else* bestemme hva som skal gjøres hvis ordet har en annen endingen enn -y.

            4.3.2.1 ta *output* og skriv hva som skal skrives ut hvis -y ikke er endingen

4.4 sende de nye formene til *<p>* ved hjelp av *document.getElementById('output').innerHTML*.

5. Lag en knapp som kan kalle funksjonen *<button onclick=' compare()'>*

```
<!DOCTYPE html>
<html lang="en">
<body>
<!-- Brukeren kan skrive inn ordet her -->
<input type="text" id="adjective">
<!-- Brukeren kan kjøre funksjonen med knappen -->
<button onclick="compare()">Compare adjectives ending in y</button>
<!-- Output vises her -->
<p id="output"></p>

<script>
//funksjonen for å skrive forskjellige adjektivens formene
function compare(){
    //definerer output
    let output;
    //definerer input og legger adjektiven inn i den
    let input= document.getElementById("adjective").value;
    //finner ut om ordet ender med -y
    if (input.endsWith("y")){
        //Lager tre forskjellige former av ordet
        output=input+"=>"+ input.slice(0,-1)+"ier"+"=>"+input.slice(0,-
1)+"iest";
    //bestemmer hva som skal gjøres hvis ordet ender ikke med -y
    } else {
        output= "Input does not end with y"
    }
    //sender det nye ordet til output <p>
    document.getElementById("output").innerHTML = output;
}
</script>
</body>
</html>
```

Eksempel output:

Figur 21

5.2.7 Simple past tense

I engelskspråket:

Regelrette verb bøyes i fortid ved å legge til -ed til grunnformen (S. Pettersen & Røkaas, 2020).

Verb som slutter med e, får bare -d i fortid (S. Pettersen & Røkaas, 2020).

I verb som slutter på konsonant+y, endrer vi y til i + ed (S. Pettersen & Røkaas, 2020).

Denne oppgaven kan endres for å passe andre språkfag også.

Oppgave:

Lag et program som skriver verb i enkelt preteritum.

Oppgaven er hentet fra (S. Pettersen & Røkaas, 2020)

Løsning:

For å lage et program som kan ta inn et ord og endre den til enkelt preteritum, må vi først lage en input boks der du kan skrive ordet inn i. Vi må også lage kode for hvordan det bestemmes hvilken ending som skal legges til ordet. Dette kan gjøres med en if- funksjonen. Vi må også lage koden for hvordan ordet blir endret. Etter det kan vi sende svar til en output tekst boks. Vi må også lage kode for å kjøre selve funksjonen, det kan gjøres med en knapp.

Algoritme:

1. Lag et HTML5 dokument.

2. Lag en *<input type='text'>* for å skrive inn ordet.

    2.1. Lag en *id='verb'* i input slik at ordet kan hentes ut til funksjonen.

3. Lag en *<p>* tekst boks.

    3.1 Lag en *id='output'* innen denne boksen slik at den nye formen av verbet som blir laget inne i funksjonen kan vises.

4. Lag en funksjon *find().*

    4.1 definer ordet *'output'* for senere

    4.2 definer ordet *'input'* og hent inn verbet fra inputen ved hjelp av *document.getElementById('verb').value*.

    4.3 lag en if- setning.

        4.3.1 først skal vi finne ut om verbet ender med -e. *.endsWith('e')*

            4.3.1.1 ta *output* og legg *input* inn i den. Legg til *+'d'* til *input*

        4.3.2 Det neste vi skal finne ut er om verbet ender med en konsonant og en -y. *input.endsWith("y")&&*

        !((input.slice(0,-1)).endsWith("a")||

        (input.slice(0,-1)).endsWith("e")||

        (input.slice(0,-1)).endsWith("i")||

        (input.slice(0,-1)).endsWith("o")||

        (input.slice(0,-1)).endsWith("u"))

            4.3.2.1 ta *output* og legg *input* inn i den. Ta vekk den siste bokstaven ved hjelp av *slice(0,-1)* og legg *+'ied'* til *input*

        4.3.3 Til slutten skal vi ved hjelp av *else* bestemme hva som skal gjøres hvis ordet ikke ender med de andre endingene.

4.3.3.1 ta *output* og legg *input* inn i den. Legg *+'ed'* til *input*

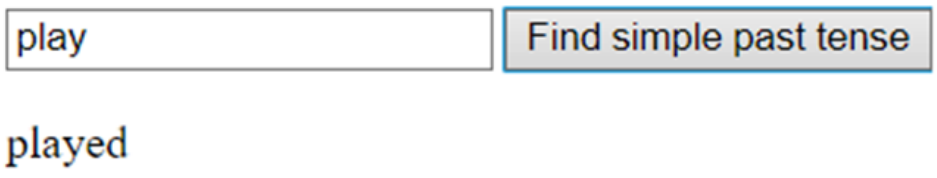
4.4 send de nye formene til *<p>* ved hjelp av *document.getElementById('output').innerHTML*.

5. Lag en knapp som kan kalle funksjonen *<button onclick=' find()'>*

```
<!DOCTYPE html>
<html lang="en">
<body>
<!-- Brukeren kan skrive inn verbet her -->
<input type="text" id="verb">
<!-- Brukeren kan kjøre funksjonen med knappen -->
<button onclick="find()">Find simple past tense</button>
<!-- Output vises her -->
<p id="output"></p>

<script>
//funksjonen for å endre verbet
function find(){
    //definerer output
    let output;
    //definerer input og legger verbet inn i den
    let input= document.getElementById("verb").value;
    //bestemmer om endingen skal være -d
    if (input.endsWith("e")){
        //legger til "d"
        output= input+"d";
    //bestemmer om endingen skal være -ied
    } else if(input.endsWith("y")&&
    //finner ut om nest siste bokstavet er en konsonant
    !((input.slice(0,-1)).endsWith("a")||
    (input.slice(0,-1)).endsWith("e")||
    (input.slice(0,-1)).endsWith("i")||
    (input.slice(0,-1)).endsWith("o")||
    (input.slice(0,-1)).endsWith("u"))){
        //tar vekk den siste bokstaven fra ordet og legger til "ied"
        output= input.slice(0,-1)+"ied"
    //bestemmer om endingen skal være -es
```

```
    } else {
        //legger til "ed"
        output= input+"ed";
    }
    //sender det nye verbet til output <p>
    document.getElementById("output").innerHTML = output;
}
</script>
</body>
</html>
```

Eksempel output:



Figur 22

Figur 23

5.2.8 First- or third- person

Denne oppgaven kan endres for å passe andre språkfag også.

Oppgave:

Lag et program som finner ut om teksten bruker første- eller tredjepersonsforteller.

Oppgaven er hentet fra (S. Pettersen & Røkaas, 2021)

Løsning:

Først er det viktig å lage en tekstboks for å legge teksten inn i. Etter det må vi lage kode for å sjekke om teksten bruker første- eller tredjepersonsforteller, dette kan gjøres med en if-funksjonen. Etter det kan vi sende svar til en output tekst boks. Vi må også lage kode for å kjøre selve funksjonen, dette kan gjøres med en knapp.

Algoritme

1. Lag et HTML5 dokument.

2. Lag en *<textarea>* for å skrive inn teksten.

    2.1. Lag en *id='inputText'* i input slik at teksten kan hentes ut til funksjonen.

3. Lag en *<p>* tekst boks.

    3.1 Lag en *id='output'* inne i denne boksen slik at svaret som blir laget inne i funksjonen kan vises.

4. Lag en funksjon *checkIfFirstPerson().*

    4.1 definer ordet *'output'* for senere

    4.2 definer ordet *'input'* og hent inn teksten fra inputen ved hjelp av *document.getElementById('inputText').value*.

    4.3 lag en if- setning.

        4.3.1 først skal vi finne ut om teksten er førsteperson. *input.includes("i ")||input.includes("you ")||input.includes("we ")*

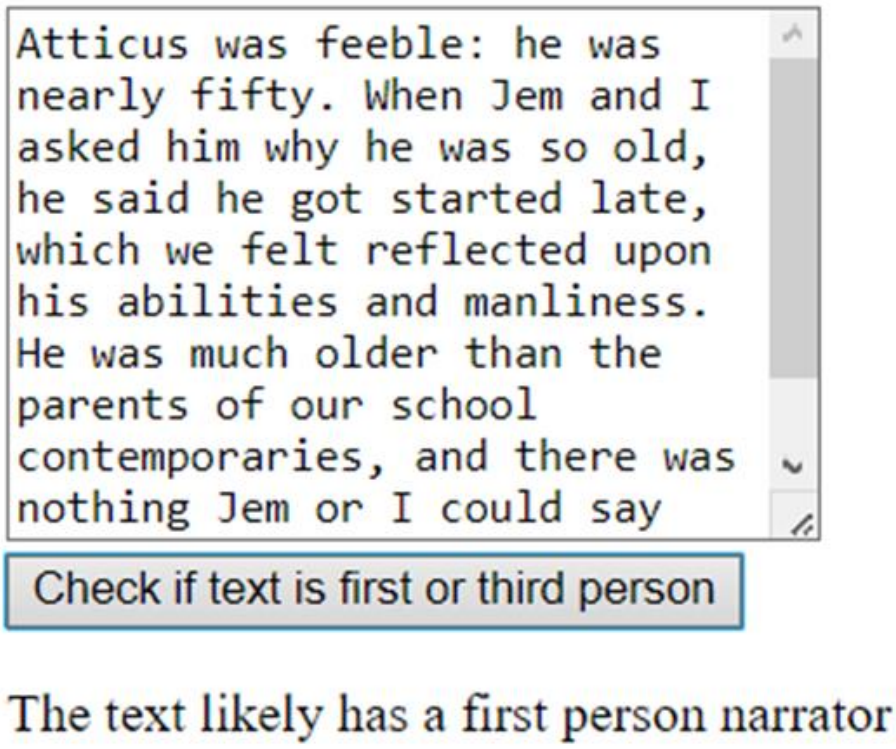
        4.3.2 dermed skal vi finne ut om teksten er tredjeperson ved hjelp av *else.*

    4.4 send svaret til *<p>* ved hjelp av *document.getElementById('output').innerHTML*.

5. Lag en knapp som kan kalle funksjonen *<button onclick=' checkIfFirstPerson()'>*

```
<!DOCTYPE html>
<html lang="en">
<body>
<!-- Brukeren kan skrive inn teksten her -->
<textarea name="Check if first person" id="inputText" cols="30"
rows="10"></textarea>
<br>
<!-- Brukeren kan kjøre funksjonen med knappen -->
<button onclick="checkIfFirstPerson()">Check if text is first or third
person</button>
<!-- Output vises her -->
<p id="output"></p>
<script>
    //funksjonen for å finne ut om teksten er førstepersonforteller
    function checkIfFirstPerson(){
        //definerer output
        let output;
        //definerer input og legger teksten inn i den
        let input= document.getElementById("inputText").value;
        //finner ut om teksten er førstepersonforteller
        if ( input.includes( "i ")||input.includes("you ")||input.includes("we
"))
        {output = "The text likely has a first person narrator"}
        //finner ut om teksten er tredjepersonforteller
        else { output = "The text likely has a third person narrator"}
        //sender svaret til output <p>
        document.getElementById("output").innerHTML = output
    }
</script>
</body>
```

Eksempel output:



Figur 24

5.2.9 Adjective to adverb

I engelskspråket:

Adverb som forteller oss hvordan noe skjer, dannes ofte ved å legge til -ly til et adjektiv (S. Pettersen & Røkaas, 2021).

I ord som ender med -y, ender vi y til i før vi legger til -ly (S. Pettersen & Røkaas, 2021).

Denne oppgaven kan endres for å passe andre språkfag også.

Oppgave:

Lag et program som gjør adjektivet om til et adverb.

Oppgaven er hentet fra (S. Pettersen & Røkaas, 2021)

Løsning:

Først er det viktig å lage en input boks for å legge ordet inn i. Etter det må vi lage kode for å sjekke om det skal legges til -ly eller -ily, dette kan gjøres med en if- funksjonen. Etter det kan vi sende svar til en output tekst boks. Vi må også lage kode for å kjøre selve funksjonen, dette kan gjøres med en knapp.

Algoritme:

1. Lag et HTML5 dokument.

2. Lag en *<input type='text'>* for å skrive inn ordet.

    2.1. Lag en *id='adjective'* i input slik at adjektivet kan hentes ut til funksjonen.

3. Lag en *<p>* tekst boks.

    3.1 Lag en *id='output'* inne i denne boksen slik at svaret som blir laget inne i funksjonen kan vises.

4. Lag en funksjon *adjectiveToAdverb().*

    4.1 definer ordet *'output'* for senere

    4.2 definer ordet *'input'* og hent inn ordet fra inputen ved hjelp av *document.getElementById('adjective').value*.

    4.3 lag en if- setning.

        4.3.1 først skal vi finne ut om ordet ender med -y. *.endsWith("y")*

            4.3.1.1 ta output og legg input inn i den. Ta vekk den siste bokstaven ved hjelp av *slice(0,-1)* og legg *+'ily'* til *input*

        4.3.2 Det neste vi skal gjøre er å finne ut om ordet skal ha endingen -ly ved hjelp av *else.*

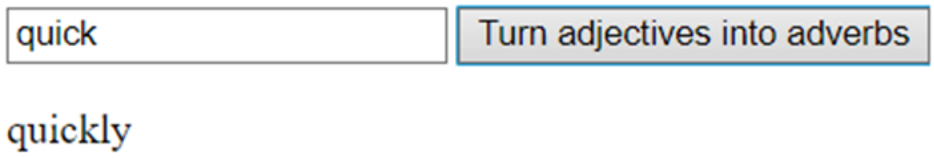
            4.3.2.1 ta output og legg input inn i den. Legg *+'ily'* til *input*

    4.4 sende svaret til *<p>* ved hjelp av *document.getElementById('output').innerHTML*.

5. Lag en knapp som kan kalle funksjonen *<button onclick=' adjectiveToAdverb()'>*

```
<!DOCTYPE html>
<html lang="en">
<body>
<!-- Brukeren kan skrive inn ordet her -->
<input type="text" id="adjective">
<!-- Brukeren kan kjøre funksjonen med knappen -->
<button onclick="adjectiveToAdverb()">Turn adjectives into adverbs</button>
<!-- Output vises her -->
<p id="outputid"></p>

<script>
//funksjonen for å endre adjektivet til et adverb
function adjectiveToAdverb(){
    //definerer output
    let output;
    //definerer input og legger ordet inn i den
    let input= document.getElementById("adjective").value;
    //finner ut om ordet skal ha -ily endingen
    if (input.endsWith("y")){
        output= input.slice(0,-1)+"ily";
    //finner ut om ordet skal ha -ly endingen
    } else {
        output= input+"ly"
    }
    //sender det nye ordet til output <p>
    document.getElementById("outputid").innerHTML = output;
}
</script>
</body>
</html>
```

Eksempel output:



Figur 25

5.3 Videregående skole

5.3.1 Random number generator

Når det kommer til språk, kan du finne tall i mange hverdagssituasjoner, alt fra økonomi til shopping. Det kan være viktig å vite hvordan man sier forskjellige tall, og i denne oppgaven skal vi lage et program som kan hjelpe deg å øve på å si disse tall.

Denne oppgaven kan brukes til andre språkfag også.

Oppgave:

Lag et program som genererer et tilfeldig tall fra 0 til 1 000 000 000. Si tallene.

Oppgaven er hentet fra (Burgess & Sørhus, 2013)

Løsning:

Først skal vi lage en output tekst boks som skal ta inn det genererte tallet. Etter det må vi lage kode for å generere tallet. Vi burde også gjøre tallet enkelt å lese. Etter det kan vi sende nummeret til en output tekst boks. Vi må også lage kode for å kjøre selve funksjonen, dette kan gjøres med en knapp.

Algoritme:

1. Lag et HTML5 dokument.

2. Lag en *<p>* tekst boks.

    2.1 Lag en *id='output'* innen denne boksen slik at nummeret som blir laget innen funksjonen kan vises.

3. Lag en funksjon *numGenerator().*

    3.1 definer ordet *'randomNum'* og generer et tilfeldig tall med *Math.Random()*

    3.2 definer ordet *'output'*, legg det genererte nummeret inn i den, og gjør den mer lesbar ved hjelp av *.toLocaleString()*

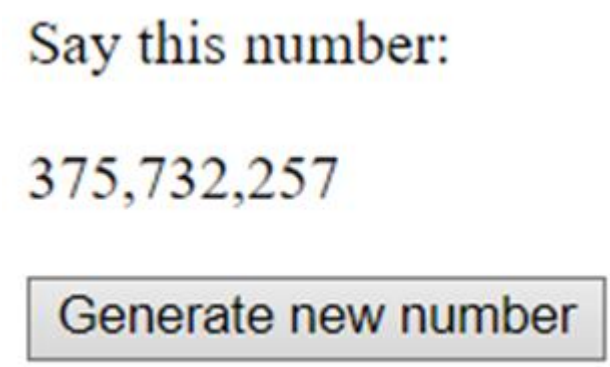
    3.3 send nummeret til *<p>* ved hjelp av *document.getElementById('output').innerHTML*.

4. Lag en knapp som kan kalle funksjonen *<button onclick=' numGenerator()'>*

```
<!DOCTYPE html>
<html lang="en">
<!-- genererer tallet når du åpner programmet -->
<body onload="numGenerator()">
    <p>Say this number:</p>
    <!-- Output vises her -->
    <p id="output"></p>
    <!-- Brukeren kan kjøre funksjonen med knappen -->
    <button onclick="numGenerator()">Generate new number</button>

    <script>
        //funksjonen for å generere tallet
        function numGenerator(){
            //lager et tilfeldig tall fra 1 til 1000000000
            let randomNum= Math.floor(Math.random() * 1000000001);
            //gjør tallet enkelt å lese. f.eks: 1000000 => 1,000,000
            let output= randomNum.toLocaleString();
            //sender det nye tallet til output <p>
            document.getElementById("output").innerHTML = output
        }
```

```
    </script>
</body>
</html>
```

Eksempel output:



Figur 26

5.3.2 Canada minorities table

Figur 27 Canada minorities table

Visible minority groups (15)	Age groups (10)									
	Total - Age groups	Under 15 years	15 to 24 years	25 to 54 years	25 to 34 years	35 to 44 years	45 to 54 years	55 to 64 years	65 to 74 years	75 years and over
Total - Population by visible minority groups	31,241,030	5,576,805	4,207,810	13,732,585	3,987,070	4,794,100	4,951,410	3,649,530	2,255,640	1,818,655
Total visible minority population [2]	5,068,090	1,145,395	785,360	2,355,455	799,245	874,930	681,280	412,775	233,060	136,055
Chinese	1,216,570	210,925	186,925	578,725	166,715	218,815	193,190	110,250	76,060	53,675
South Asian [3]	1,262,865	305,215	181,410	577,800	216,790	208,060	152,945	108,025	61,550	28,865
Black	783,800	221,660	130,015	328,750	120,230	124,570	83,950	58,535	29,805	15,030
Filipino	410,695	89,785	53,885	203,715	63,330	77,900	62,490	37,195	16,685	9,435
Latin American	304,245	59,915	51,885	157,675	56,995	57,625	43,055	21,960	8,360	4,450
Southeast Asian [4]	239,935	55,360	38,270	116,900	37,540	42,865	36,495	15,265	8,445	5,700
Arab	265,550	69,650	40,980	126,175	47,405	48,605	30,160	16,405	8,130	4,210
West Asian [5]	156,695	30,845	29,190	77,450	25,050	28,965	23,435	11,125	5,360	2,725
Korean	141,890	27,275	28,945	66,650	20,965	23,435	22,245	10,575	5,745	2,700
Japanese	81,300	14,900	10,295	35,895	13,165	13,700	9,030	8,580	6,005	5,630
Visible minority, n.i.e. [6]	71,420	14,305	11,375	34,835	12,485	12,440	9,910	6,265	2,955	1,680
Multiple visible minority [7]	133,120	45,550	22,180	50,885	18,570	17,940	14,370	8,595	3,955	1,950
Not a visible minority [8]	26,172,935	4,431,410	3,422,455	11,377,130	3,187,830	3,919,165	4,270,130	3,236,755	2,022,580	1,682,600

Kilde: (Statistics Canada, 2020)

Denne oppgaven kan endres for å passe andre språkfag også.

Oppgave:

Se på følgende statistikk som omhandler synlige minoriteter i Canada etter aldersgruppe, og svar spørsmål:

A) Hvor stor prosentandel av befolkningen i Canada er sammensatt av minoriteter?

B) Hvilken er den største minoriteten i Canada?

C) Hvilken er den nest største?

D) Hvor mange prosent av befolkningen er mellom 15 og 24 år?

E) Hvor stor prosentandel av den totale minoritetsbefolkningen utgjør kinesere?

Oppgaven er hentet fra (Burgess & Sørhus, 2013)

Løsning:

Først skal vi lage 5 output tekst bokser for alle fem deloppgaver. Etter det skal vi lage en funksjon for å løse a), d), og e). Funksjonen skal regne ut prosentandelen. Dermed skal vi kalle funksjonen for a), d), og e) og sende svar til output. Til b) skal vi skrive kode for å finne den største, og for c) skal vi ta vekk svaret fra b) og finne den største igjen.

Algoritme

1. Lag et HTML5 dokument.

2. Lag 5 *<p>* tekst bokser.

2.1 Lag en *id='a'* til *id='e'* innen disse boksene slik at svarene kan legges inn i disse.

3. Lag en funksjon *percent(num1, num2).*

    3.1 finn prosenten av to numre ved *(num2/num1)*100*

4. definer a til e.

    4.1 for a) skal du ta "Total - Population by visible minority groups" og "Total visible minority population" og legge disse tall inn i funksjonen *percent.* Send svar til tekstboksen *<p>*

    4.2 for b) skal du skrive alle minoriteter inn i *Math.max()* funksjonen for å finne den største. Send svar til tekstboksen *<p>*

    4.3 for c) skal du skrive alle minoriteter, uten den du fikk fra b), inn i *Math.max()* funksjonen for å finne den største. Send svar til tekstboksen *<p>*

    4.4 for d) skal du ta "Total - Population by visible minority groups" for alle aldere og for 15 til 24 aldere og legge disse tall inn i funksjonen percent. Send svar til tekstboksen <p>

4.1 for a) skal du ta “Total visible minority population” og “Chinese” og legge disse tall inn i funksjonen *percent.* Send svar til tekstboksen *<p>*

```
<!DOCTYPE html>
<html lang="en">
<body>
<!-- Output vises her -->
<p id="a"></p>

<p id="b"></p>

<p id="c"></p>

<p id="d"></p>

<p id="e"></p>
    <script>
    //funksjonen for å regne prosentandelen
    function percent(num1, num2){
        return (num2/num1)*100;
    }

    //a)
    //bruke funksjonen for å finne prosentandelen
    let a = percent(31241030, 5068095)
    //sende svar til <p>
    document.getElementById("a").innerHTML = a

    //b)
    //finne den største
    let b = Math.max(1262865, 1216565, 783795, 410700, 304245, 265550, 239935,
156695, 141890, 81300)
    //sende svar til <p>
    document.getElementById("b").innerHTML = b

    //c)
```

```
    //finne den største
    let c = Math.max(1216565, 783795, 410700, 304245, 265550, 239935, 156695,
141890, 81300)
    //sende svar til <p>
    document.getElementById("c").innerHTML = c

    //d)
    //bruke funksjonen for å finne prosentandelen
    let d = percent(31241030, 4207815)
    //sende svar til <p>
    document.getElementById("d").innerHTML = d

    //e)
    //bruke funksjonen for å finne prosentandelen
    let e = percent(5068095, 1216565)
    //sende svar til <p>
    document.getElementById("e").innerHTML = e
    </script>
</body>
</html>
```

Eksempel output:

16.222560523772746

1262865

1216565

13.468874105623277

24.00438429034973

Figur 28

5.3.3 Currency conversion

På tiden av skriving, er konversjonen fra USD til NOK 10.6006

Denne oppgaven kan endres for å passe andre språkfag også.

Oppgave:

Se på følgende produkter:

1. Apple iPod nano 4GB originalt kostet $149, redusert 10%
2. PC spill originalt kostet $29.99, redusert 20%
3. Digitalt kamera originalt kostet $249.99, redusert 25%

Finn ut redusert pris og konverter USD til NOK.

Oppgaven er hentet fra (Burgess & Sørhus, 2013)

Løsning:

Først skal vi lage en input boks for å skrive inn prisen, og en input boks for å skrive hvor mye prosent den ble redusert. Lag også en knapp for å kjøre funksjonen. Lag også en input boks for å konvertere USD til NOK. Lag en knapp for å kjøre funksjonen. Så skal vi lage funksjonen for å redusere prisen, og en funksjon for å konvertere USD til NOK, og sende svar til output tekst boksene.

Algoritme:

1. Lag et HTML5 dokument.

2. Lag en *<input type='text'>* for å skrive inn prisen med en *id='price'*.

3. Lag en *<input type='text'>* for å skrive inn prosenten med en *id='percent'*.

4. Lag en *<p>* tekst boks med en *id='output1'* innen denne boksen.

5. Lag en knapp som kan kalle funksjonen *<button onclick=' reducedPrice()'>*

6. Lag en *<input type='text'>* for å skrive inn USD med en *id='usd'*

7. Lag en <p> tekst boks med en id='output1' innen denne boksen.

8. Lag en knapp som kan kalle funksjonen *<button onclick=' exchangeRate()'>*

9. Lag en funksjon *reducedPrice()*

    9.1. definer ordet *'num1'* og hent inn prisen fra inputen ved hjelp av *Number(document.getElementById('price').value)*.

    9.2 definer ordet *'percent'* og hent inn verbet fra inputen ved hjelp av *Number(document.getElementById('percent').value)*.

    9.3 definer ordet *'sum'* og regn ut redusert pris. *num1 *((100-percent)/100)*

    9.4 send svaret til *<p>* ved hjelp av *document.getElementById('output1').innerHTML*

10. Lag en funksjon *exchangeRate()*

    10.1. definer ordet *'num2'* og hent inn USD fra inputen ved hjelp av *document.getElementById('usd').value*.

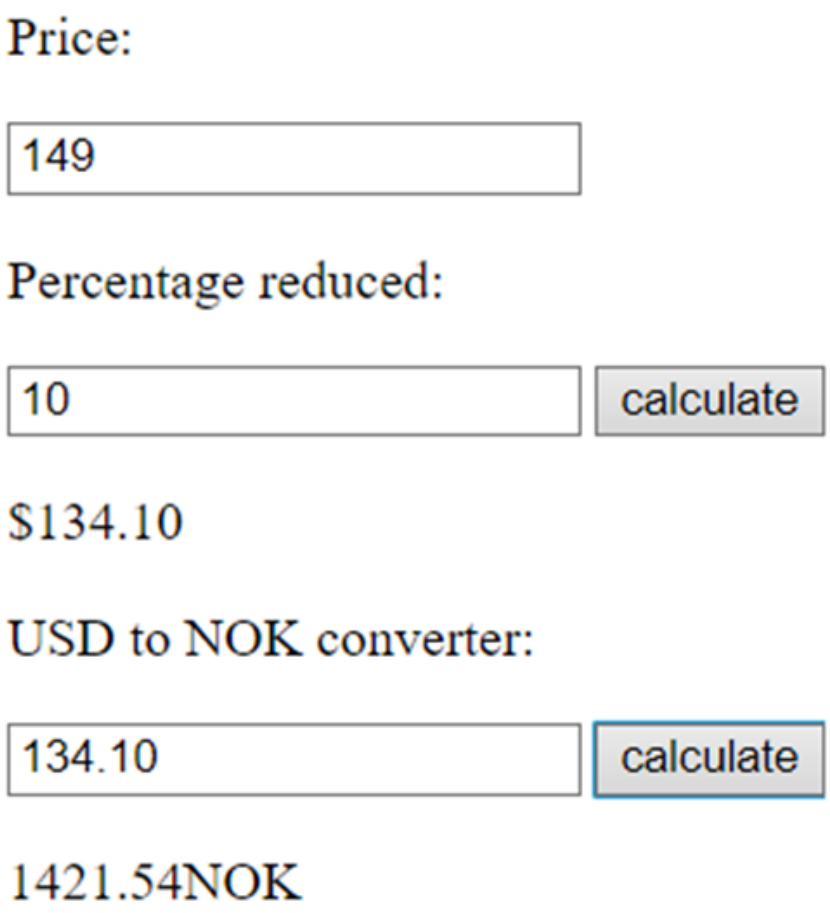
    10.2. definer ordet *'total'* og konverter til NOK. *num2 *10.6006*

    10.3. send svaret til <p> ved hjelp av *document.getElementById('output2').innerHTML*

```
<!DOCTYPE html>
<html lang="en">
<body>
<p>Price:</p>
<!-- Brukeren kan skrive inn prisen her -->
<input type="text" id="price">
<p>Percentage reduced:</p>
<!-- Brukeren kan skrive inn prosenten her -->
<input type="text" id="percent">
<!-- Brukeren kan kjøre funksjonen med knappen -->
<button onclick="reducedPrice()">calculate</button>
<!-- Output vises her -->
```

```
<p id="output1"></p>

<p>USD to NOK converter:</p>
<!-- Brukeren kan skrive inn USD her -->
<input type="text" id="usd">
<!-- Brukeren kan kjøre funksjonen med knappen -->
<button onclick="exchangeRate()">calculate</button>
<!-- Output vises her -->
<p id="output2"></p>

<script>
//funksjonen for å redusere prisen
function reducedPrice() {
//definerer num1 og legger prisen inn i den
 let num1 = Number(document.getElementById("price").value);
 //definerer percent og legger prosenten inn i den
 let percent = Number(document.getElementById("percent").value);
//definerer sum og regner ut reduksjonen
 let sum = num1 *((100-percent)/100);
 //sender redusert pris til output <p>
 document.getElementById("output1").innerHTML = "$"+sum.toFixed(2);
}
//funksjonen for å konvertere USD til NOK
function exchangeRate(){
//definerer num2 og legger USD inn i den
let num2 = document.getElementById("usd").value;
//definerer total og konverterer til NOK
let total= num2 *10.6006;
//sender konverteringen til output <p>
document.getElementById("output2").innerHTML = total.toFixed(2)+"NOK";
}
</script>
</body>
</html>
```

Eksempel output:



Figur 29

5.3.4 Quiz

Denne oppgaven kan endres for å passe andre språkfag også.

Oppgave:

Lag et program som fungerer som en flervalgsquiz. Spørsmålene i quizen skal handle om historie/litteraturhistorie/språk i gjeldende språkfag. Gi medeleven quizen slik at han/hun kan prøve å løse den.

Løsning:

For å lage et slikt program må man lage et HTML-dokument med en JavaScript-del i form av et <script>. I HTML dokumentet må man lage et div-element som kan inneholde alle spørsmålene i form av label-elementer, og svarsalternativene som options-elementer. Man må også kode inn en knapp som skal klikkes på for å sjekke om svarene er rette eller gale, i tillegg til et element som skal vise fram svaret.

I Javascript-delen må man kode inn en variabel for hvert spørsmål og svar-knappen fra HTML-delen. Til slutt skal det være med en onclick-funksjon som bekrefter om svarene er rett eller ikke basert på hvilke av options-elementene som er valgt for hvert spørsmål. Funksjonen skal også gi en tekstlig output til svar-elementet som bekrefter om brukeren klarte quizen eller ikke.

Algoritme:

1. Begynn først med å finne ut hvilke spørsmål og svarsalternativer du vil ha med i quizen

2. Lag et standard HTML5 dokument

3. Lag et * * element inne i <body> delen av dokumentet

    3.1. Lag et *<label>* element hvor du skriver inn et av spørsmålene til quizen

    3.2. Plasser deretter inn et *<select>* element under *<label>* elementet og skriv inn en id til elementet. Et eksempel kan være *<option id="spm1">*

        3.2.1 Skriv inn alle svarsalternativene i form av *<option>* elementer i *<select>* elementet

        3.2.2. I tillegg til å skrive svarsalternativet som normal tekst, må man skrive dem som en *"value"* i *<option>* elementet

    3.3. For å få linjeavstand mellom hvert spørsmål, kan man skrive inn et *
* element i mellom hvert *<select>* element

    3.4 Repeter prosessen fra 3.1-3.3 til hvert spørsmål du skal ha med i quizen

4. Lag et *<button>* element med en id-en *"knapp"*

5. Lag et tekst-element som for eksempel *<p>* med id-en *"svar"* som skal vise fram bekreftelse på om quizen blir løst korrekt eller ikke

6. Lag en Javascript del i form av elementet *<script>* inne i *<body>* elementet under resten av elementene som er skrevet

6.1. Begynn med å kode inn variabler for hvert spørsmål fra *<select>* elementene som for eksempel: *"let spm1" "spm2"* osv

6.1.1. Referer til id-en i hvert *<select>* element med funksjonen *"document.getElementById("select-id")"*

6.2. Lag også en variabel til *<button>* elementet hvor du refererer til id-en på samme måte som i 6.1.1.

6.3. Lag en *"Onclick"* funksjon til *<button>* elementet

6.3.1. Begynn med å kode inn en if-funksjon hvor du inkluderer variablene til hvert spørsmål og korrekt svarsalternativ som ble skrevet i *"value"* slik:

```
if (spm1.options[spm1.selectedIndex].text == "Realismen"
    && spm2.options[spm2.selectedIndex].text == "Forfatter"
    && spm3.options[spm3.selectedIndex].text == "Amtmans Døtre")
```

6.3.1.1 Det må skrives et *"&&"* tegn mellom hvert spørsmål som blir inkludert i if-funksjonen

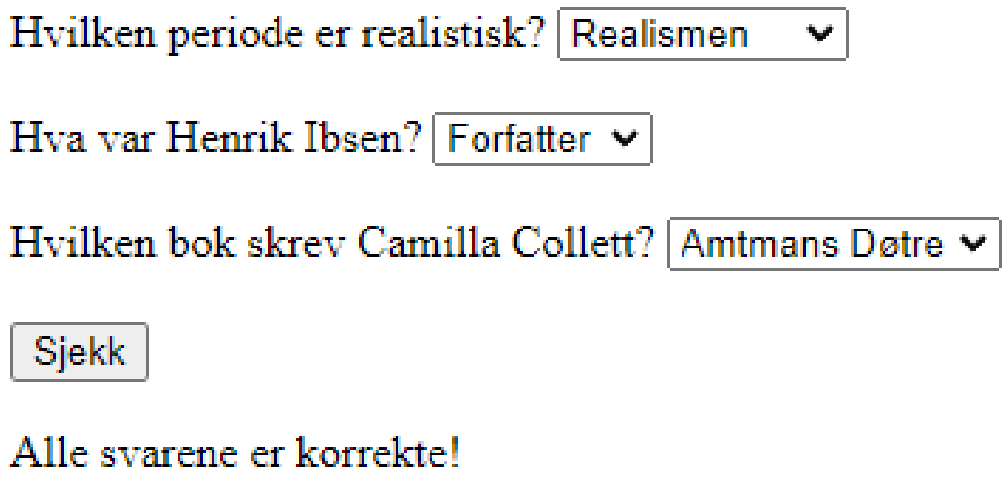
6.3.2. I slutten av if-delen må man skrive inn svaret man vil skal bli gitt til *<p>* elementet når brukeren svarer rett på quizen. Dette skrives med funksjonen: *document.getElementById("svar").innerHTML = "Alle svarene er korrekte!";*

6.3.3. I else-delen må man skrive inn svaret man vil skal bli gitt til *<p>* elementet når brukeren svarer feil på quizen. Dette skrives med funksjonen: [*{ document.getElementById("svar").innerHTML = " Feil! Refresh siden og prøv på nytt"; }*

```html
<!DOCTYPE html>
<html lang="en">
<body>
    <div>
        <label>Hvilken periode er realistisk?</label>
        <select id="spm1">
            <option value="Velg">Velg</option>
            <option value="Ingen">Ingen</option>
            <option value="Realismen">Realismen</option>
            <option value="Naturalismen">Naturalismen</option>
            <option value="Romantikken">Romantikken</option>
        </select><br></br>

        <label>Hva var Henrik Ibsen?</label>
        <select id="spm2">
            <option value="Velg">Velg</option>
            <option value="Hoteleier">Hoteleier</option>
            <option value="Kanin">Kanin</option>
            <option value="Prest">Prest</option>
            <option value="Forfatter">Forfatter</option>
        </select><br></br>

        <label>Hvilken bok skrev Camilla Collett?</label>
        <select id="spm3">
            <option value="Velg">Velg</option>
            <option value="Harry Potter">Harry Potter</option>
            <option value="Amtmans Døtre">Amtmans Døtre</option>
            <option value="Sult">Sult</option>
            <option value="Et Dukkehjem">Et Dukkehjem</option>
        </select><br></br>
    </div>

    <button type="button" id="knapp">Sjekk</button>
    <p id="svar"></p>

    <script>
        let spm1 = document.getElementById("spm1");
        let spm2 = document.getElementById("spm2");
        let spm3 = document.getElementById("spm3");
        let knapp = document.getElementById("knapp");

        knapp.onclick = function() {
            if (spm1.options[spm1.selectedIndex].text == "Realismen"
            && spm2.options[spm2.selectedIndex].text == "Forfatter"
            && spm3.options[spm3.selectedIndex].text == "Amtmans Døtre")
            {
```

```
                document.getElementById("svar").innerHTML = "Korrekt!
Realismen er realistisk, Henrik Ibsen er en berømt norsk forfatter, og Camilla
Collett skrev Amtmans Døtre";
            } else {
                document.getElementById("svar").innerHTML = "Feil! Refresh
siden og prøv på nytt";
            }

        }
    </script>
</body>
</html>
```

Eksempel output:



Figur 30

5.3.5 Miles and fahrenheit conversion

Denne oppgaven kan endres for å passe andre språkfag også.

Oppgave:

Se på følgende fakta som omhandler noen av Lynne Cox sine prestasjoner og konverter miles til kilometer og Fahrenheit til Celsius for å gjøre teksten mer forståelig. En mil er 1.609 kilometer.

Since swimming the English Channel (33 miles) in 1972, Lynne Cox has continued to look for and overcome new challenges. In 1975 Cox swam the 10-mile Cook Strait in New Zealand in water temperatures of only 50 degrees F°. In 1987 she swam the Bering Strait in waters which averaged 40 degrees. When she swam in the freezing waters of Antarctica, she was in the water for 25 minutes and swam a distance of 1.06 miles. Scientists are amazed at her perseverance as most people would have frozen within 5 minutes!

Oppgaven er hentet fra (Burgess & Sørhus, 2013)

Løsning:

Først skal vi lage en input boks for konvertering av mil til km. Lag også en knapp for å kjøre funksjonen, og en output boks. Lag også en input boks for å konvertere fahrenheit til celsius. Lag en knapp for å kjøre funksjonen, og en output boks. Så skal vi lage funksjonen for å konvertere mil til km, og en funksjon for å konvertere fahrenheit til celsius. Send disse svar til output.

Algoritme:

1. Lag et HTML5 dokument.

2. Lag en *<input type='text'>* for å skrive inn mil med en *id='miles'*.

4. Lag en knapp som kan kalle funksjonen *<button onclick=' milesToKm()'>*

5. Lag en *<p>* tekst boks med en *id='output1'*

6. Lag en *<input type='text'>* for å skrive inn fahrenheit med en *id='f'*

7. Lag en knapp som kan kalle funksjonen *<button onclick=' fToC()'>*

8. Lag en <p> tekst boks med en *id='output2'*

9. Lag en funksjon *milesToKm()*

    9.1. definer ordet *'num1'* og hent inn mil fra inputen ved hjelp av *document.getElementById('miles').value*.

9.2 definer ordet *'total'* og regn ut konverteringen. *num1 *1.609*

9.3 send svaret til *<p>* ved hjelp av *document.getElementById('output1').innerHTML*

10. Lag en funksjon *fToC()*

10.1. definer ordet *'num2'* og hent inn fahrenheit fra inputen ved hjelp av *document.getElementById('f').value*.

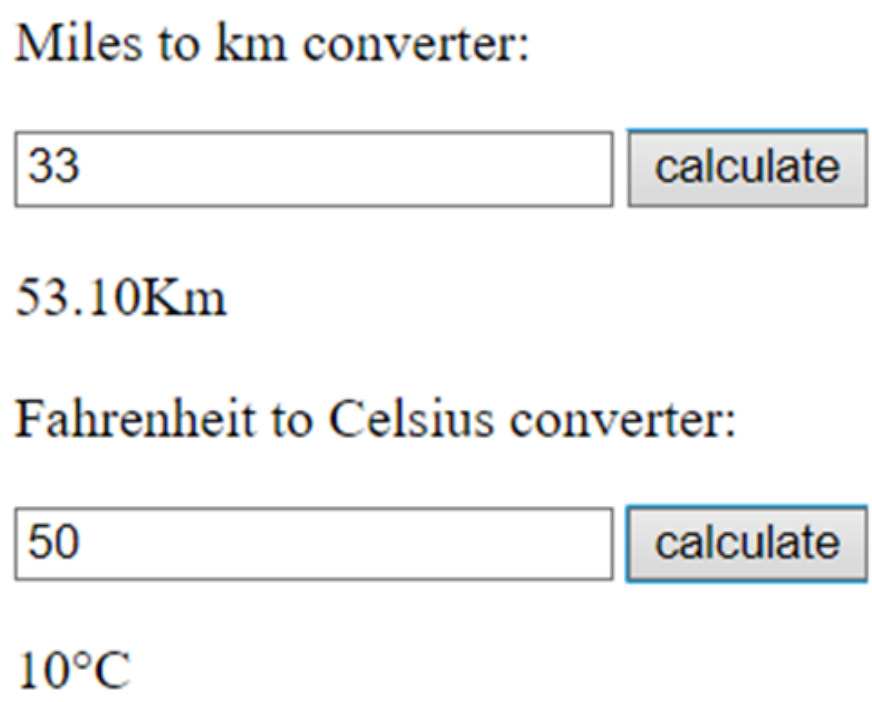
10.2. definer ordet *'total'* og regn ut konverteringen. *(num2 -32)/1.8*

10.3. send svaret til <p> ved hjelp av *document.getElementById('output2').innerHTML*

```
<!DOCTYPE html>
<html lang="en">
<body>
<p>Miles to km converter:</p>
<!-- Brukeren kan skrive inn mil her -->
<input type="text" id="miles">
<!-- Brukeren kan kjøre funksjonen med knappen -->
<button onclick="milesToKm()">calculate</button>
<!-- Output vises her -->
<p id="output1"></p>

<p>Fahrenheit to Celsius converter:</p>
<!-- Brukeren kan skrive inn fahrenheit her -->
<input type="text" id="f">
<!-- Brukeren kan kjøre funksjonen med knappen -->
<button onclick="fToC()">calculate</button>
<!-- Output vises her -->
<p id="output2"></p>

<script>
//funksjonen for å konvertere mil til km
function milesToKm(){
//definerer num1 og legger mil inn i den
let num1 = document.getElementById("miles").value;
//definerer total og regner konverteringen
let total= num1 *1.609;
```

```
//sender konverteringen til output <p>
document.getElementById("output1").innerHTML = total.toFixed(2)+"Km";
}
//funksjonen for å konvertere fahrenheit til celsius
function fToC(){
//definerer num2 og legger fahrenheit inn i den
let num2 = document.getElementById("f").value;
//definerer total og regner konverteringen
let total= (num2 -32)/1.8;
//sender konverteringen til output <p>
document.getElementById("output2").innerHTML = total+"°C";
}
</script>
</body>
</html>
```

Eksempel output:



Figur 31

5.3.6 Alphabetical order

Denne oppgaven kan endres for å passe andre språkfag også.

Oppgave:

Nedenfor er en liste over noen av de amerikanske urfolkstammene eller nasjonene som finnes i USA. Sett listen i alfabetisk rekkefølge.

Navajo, Apache, Cherokee, Seminole, Pueblo, Mohawk, Shoshone, Hopi, Sioux, Cheyenne, Seneca, Comanche, Blackfeet, Chippewa

Oppgaven er hentet fra (Burgess & Sørhus, 2013)

Løsning:

Først skal vi lage en output boks for å skrive ut svar. Det neste vi skal gjøre er å lage en array som inneholder alle stammene og nasjonene. Så skal vi sortere disse stammene, og sende det til output boksen.

Algoritme:

1. Lag et HTML5 dokument.

2. Lag en *<p>* tekst boks med en *id='output'*

3. Definer arrayen *names* og skriv inn alle navnene

4. Sorter navnene med *names.sort*

5. Send svaret til <p> ved hjelp av *document.getElementById('output').innerHTML*

```
<!DOCTYPE html>
<html lang="en">
<body>
    <!-- Output vises her -->
    <p id="output"></p>
    <script>
```

```
    //arrayen til navnene
    let names = ["Navajo", "Apache", "Cherokee", "Seminole", "Pueblo",
"Mohawk", "Shoshone", "Hopi", "Sioux", "Cheyenne", "Seneca", "Comanche",
"Blackfeet", "Chippewa"];
    //sorterer navnene alfabetisk
    names.sort();
    //sender arrayen til output <p>
    document.getElementById("output").innerHTML = names;
    </script>
</body>
</html>
```

Eksempel output:

Apache,Blackfeet,Cherokee,Cheyenne,Chippewa,Comanche,Hopi,Mohawk,Navajo,Pueblo,Seminole,Seneca,Shoshone,Sioux

Figur 32

5.3.7 Largest to smallest

Denne oppgaven kan endres for å passe andre språkfag også.

Oppgave:

Sett følgende liste i rekkefølge fra størst til minste. Listen viser befolkningen til de åtte største amerikanske urfolkstammene. (2010 folketelling)

Apache	111,810
Blackfeet	105,304
Cherokee	819,105
Chippewa	170,742
Choctaw	195,764
Mexican American Indian	175,494
Navajo	332,129
Sioux	170,110

Oppgaven er hentet fra (Burgess & Sørhus, 2013)

Løsning:

Først skal vi lage en output boks for å skrive ut svar. Det neste vi skal gjøre er å lage en array som inneholder alle stammene. Så skal vi sortere disse stammene, og sende det til output boksen.

Algoritme:

1. Lag et HTML5 dokument.

2. Lag en *<p>* tekst boks med en *id='output'*

3. Definer arrayen *tribes* og skriv inn alt befolkningstall

4. Sorter stammene med *tribes.sort(function(a, b) {return b - a; });*

5. Send svaret til <p> ved hjelp av *document.getElementById('output').innerHTML*

```
<!DOCTYPE html>
<html lang="en">
<body>
<!-- Output vises her -->
<p id="output"></p>
    <script>
    //arrayen til stammene
    let tribes = [111810, 105304, 819105, 170742, 195764, 175494, 332129,
170110];
    //sorterer stammene størst til minst
    tribes.sort(function(a, b) {
    //hjelper å sortere nummerisk og ikke alfabetisk
    return b - a;
    });
    //sender arrayen til output <p>
    document.getElementById("output").innerHTML = tribes;
    </script>
</body>
```

```
</html>
```

Eksempel output:

> 819105,332129,195764,175494,170742,170110,111810,105304

Figur 33

5.3.8 Informal language

Denne oppgaven kan brukes i andre språkfag også.

Oppgave:

Lag et program som bestemmer om teksten er formell eller uformell

Oppgaven er hentet fra (Underwood et al., 2021).

Løsning:

Først skal vi lage en textarea boks for å skrive inn teksten. Så skal vi lage en knapp for å kjøre funksjonen, og en output tekst boks. Videre skal vi lage en funksjon som, ved bruk av if-funksjonen, finner ut om teksten bruker første- og andrepersonspronomen og om det er bruk av sammentrekninger. Det er mulighet til å utvide koden med enda flere regler.

Algoritme:

1. Lag et HTML5 dokument.

2. Lag en *textarea* input boks med *id='inputId'*

3. Lag en knapp for å kjøre funksjonen

4. Lag en <p> output tekst boks med *id='output'*

5. Lag funksjonen *checkIfInformal()*

5.1 Definer ordet *output* for senere

5.2 Definer *input* og sett teksten inn i den

5.3 Lag en if- funksjon

5.3.1 finn ut om teksten har første og andrepersonspronomen ved hjelp av *.includes()*

5.3.2 finn ut om teksten har sammentrekninger ved hjelp av *.includes()*

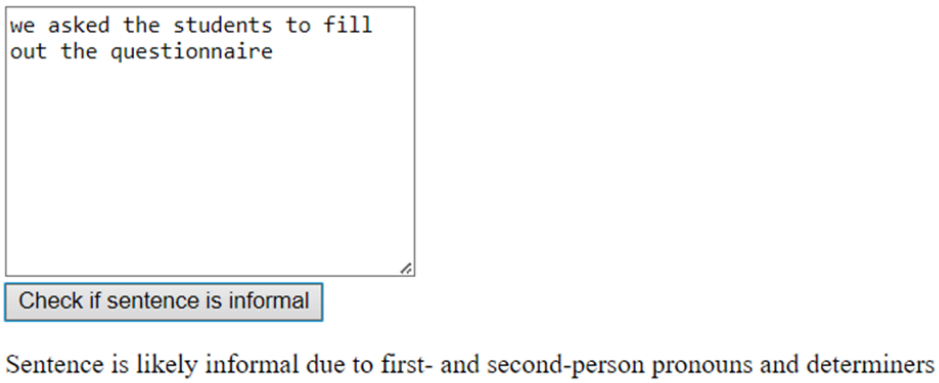
5.3.3 bruk *else* for å bestemme hva som skal gjøres hvis disse finnes ikke

5.4 Send outputen til <p> tekst boks ved hjelp av *document.getElementById("output").innerHTML*

```
<!DOCTYPE html>
<html lang="en">
<body>
<!-- Brukeren kan skrive inn teksten her -->
<textarea name="Check if informal" id="inputId" cols="30"
rows="10"></textarea>
<br>
<!-- Brukeren kan kjøre funksjonen med knappen -->
<button onclick="checkIfInfromal()">Check if sentence is informal</button>
<!-- Output vises her -->
<p id="output"></p>
<script>
    //funksjonen for å finne ut om setningen er uformell
    function checkIfInfromal(){
        //definerer output
        let output;
        //definerer input og legger teksten inn i den
        let input= document.getElementById("inputId").value;
        //finner ut om det finnes i, you, eller we i teksten og hva som skal
gjøres hvis det skjer
```

```
        if ( input.includes( "i ")||input.includes("you ")||input.includes("we
"))
        {output = "Sentence is likely informal due to first- and second-person
pronouns and determiners"}
        //finner ut om det finnes sammentrekninger i teksten og hva som skal
gjøres hvis det skjer
        else if (input.includes( "'"))
        {output = "Sentence is likely informal due to contractions"}
        //bestemmer hva som skal gjøres hvis ingenting finnes
        else { output = "Sentence is likely formal, but you should still check
for slang words, abbreviations, and the structure of the sentence to make
sure"}
        document.getElementById("output").innerHTML = output
    }
</script>
</body>
</html>
```

Eksempel output:



Figur 34

5.3.9 Tidslinje

Denne oppgaven kan endres for å passe andre språkfag også.

Oppgave:

Lag et program som viser fram en tidslinje av de viktigste litterære tidene gjennom det gitte språkfagets historie.

Løsning:

For å lage et program som viser fram en tidslinje, må man lage en rekke elementer i HTML og style elementene i CSS slik at de vil få utseende og formateringen til en tidslinje. I HTML må man lage et element som skal inneholde alle elementene til tidslinjen. Man lager da en rekke elementer med "class=container" som enten skal være høyre eller venstre kontainere på tidslinjen. Inne i hver kontainer skal man kode inn enda et element som får "class=text-box". Der skal man skrive inn tekst elementer som skal være selve innholdet om litterær perioden man beskriver for den gitte konteineren på tidslinjen.

I tillegg til å lage elementer i HTML må man som sagt style elementene ved hjelp av CSS slik at de skal se ut som en tidslinje. Man styler da posisjonen, fargen, størrelsen og outlinen på containerne, text-boxene og tidslinjen for å få det til å se korrekt ut.

Algoritme:

1. Lag et HTML5 dokument

2. Lag et * * element i *<body>* med *"class="timeline""*

2.1. Lag et * * element inne i *"timeline"* div-en som har *"class= container left-container"* eller *"class= container right-container"*. Det spiller ingen rolle hvilken som blir valgt til den første *"container"* div-en

2.1. Lag et * * element inne i *"container"* div-en som har *"class="text-box"*

2.1.1. Inne i *"text-box"* div-en koder man inn et *<h2>* element hvor man skriver navnet på litteratur perioden som skal dekkes i boksen

2.1.2. Under *<h2>* elementet lager man et *<small>* element hvor man skriver hvilken tidsperiode den gitte perioden tok sted. Til Realismen kan man for eksempel skrive "1830 – 1890"

2.1.3. Til slutt skriver man selve teksten og innholdet om perioden i et *<p>* element under *<small>* elementet

2.2. Repeter prosessen i 2.1. for hver periode som skal være med i tidslinjen.

2.2.1. Det er viktig å huske at man må veksle mellom å kode inn *"class= container left-container"* og *"class= container right-container"* for hver nye *"container"* * * man legger til

3. Lag et *<style>* element under *<body>* elementet

3.1. Lag en style for **{* hvor du koder:

{

margin: 0;

padding: 0;

box-sizing: border-box;

}

3.2. Lag en style for body med "background-color: valgfri farge" slik at bakgrunnsfargen på hele siden utenom selve tidslinjen blir slik du ønsker den

3.3. Lag style for *.timeline* hvor du koder:

{

position: relative;

max-width: 1200px;

margin 100px auto;

}

3.4. Lag style for *.container* hvor du koder:

{

padding: 10px 50px;

position: relative;

width: 50%;

}

3.5. Lag style for *.text-box* hvor du koder:

{

padding: 20px 30px;

background: valgfri farge;

position: relative;

border-radius: 6px;

font-size: 15px;

}

3.6. Lag en style for *.left-container* med *“left: 0:”* slik at kontainere med *“class=container left-container""* blir plassert korrekt på venstre side av tidslinjen

3.7. Lag en style for *.right-container* med *“left: 50%:”* slik at kontainere med *“class=container right-container""* blir plassert korrekt på høyre side av tidslinjen

3.8 Lag en style for *.timeline::after* hvor du koder:

{

content: '';

position: absolute;

width: 6px;

height: 100%;

background: #fff;

top: 0;

left: 50%;

margin-left: -3px;

z-index: -1;

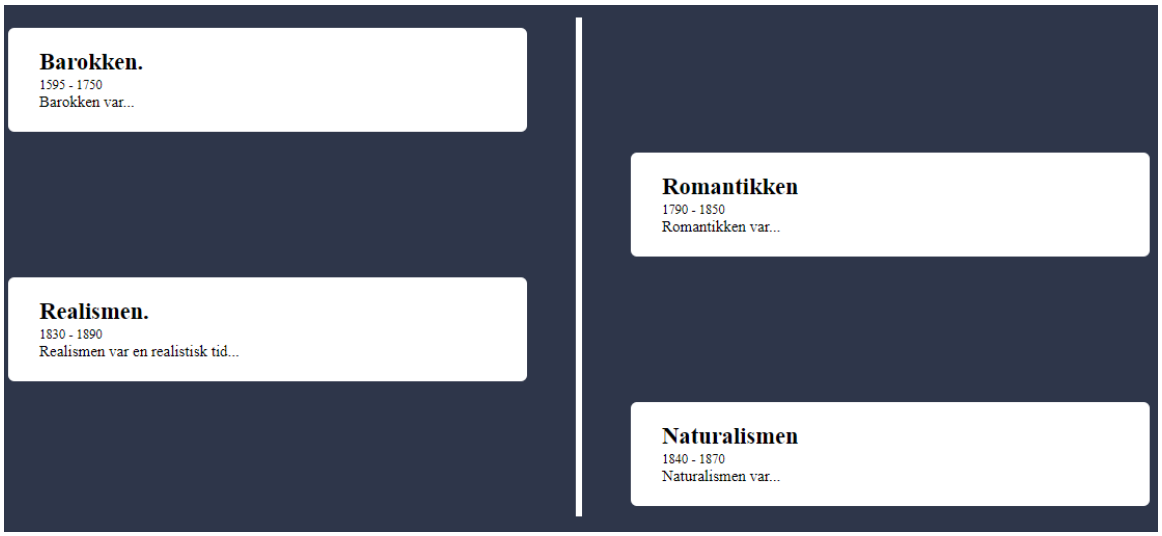
}

```
<!DOCTYPE html>
<html lang="en">
<body>
    <div class="timeline">
        <div class="container left-container">
            <div class="text-box">
                <h2>Barokken.</h2>
                <small>1595 - 1750</small>
                <p>Barokken var...</p>
            </div>
        </div>
        <div class="container right-container">
            <div class="text-box">
                <h2>Romantikken</h2>
                <small>1790 - 1850</small>
                <p>Romantikken var...</p>
            </div>
        </div>
        <div class="container left-container">
            <div class="text-box">
                <h2>Realismen.</h2>
                <small>1830 - 1890</small>
                <p>Realismen var en realistisk tid...</p>
            </div>
        </div>
        <div class="container right-container">
            <div class="text-box">
                <h2>Naturalismen</h2>
                <small>1840 - 1870</small>
                <p>Naturalismen var...</p>
```

```
            </div>
        </div>
    </div>
</body>

<style>
    *{
        margin: 0;
        padding: 0;
        box-sizing: border-box;
    }
    body{
        background-color: #2e364a;
    }
    .timeline {
        position: relative;
        max-width: 1200px;
        margin: 100px auto;
    }
    .container{
        padding: 10px 50px;
        position: relative;
        width: 50%;
    }
    .text-box{
        padding: 20px 30px;
        background: white;
        position: relative;
        border-radius: 6px;
        font-size: 15px;
    }
    .left-container{
        left: 0;
    }
    .right-container{
        left: 50%;
    }
    .timeline::after{
        content: '';
        position: absolute;
        width: 6px;
        height: 100%;
        background: #fff;
        top: 0;
        left: 50%;
        margin-left: -3px;
        z-index: -1;
    }
```

```
</style>
</html>
```

Eksempel output:



Figur 35

5.3.10 Check for repetitions

"Figures of speech are a set of literary techniques that are used primarily for their effect. They are most associated with poetic and literary language, but they are also found in language that is intended to persuade or impress an audience." Talefigurer kan bli klassifisert inn i to: troper og ordninger. Vanlige ordninger kan være: allitterasjon, assonans, utelatelse og repetisjon (Underwood et al., 2021).

Denne oppgaven kan brukes i andre språkfag også.

Oppgave:

Lag et program som kan finne alle repetisjonene i en sang.

Oppgaven er hentet fra (Underwood et al., 2021)

Løsning:

Først skal vi lage en textarea boks for å skrive inn teksten. Så skal vi lage en knapp for å kjøre funksjonen, og en output tekst boks. Videre skal vi gjøre teksten om til en array, og lage en funksjon som, ved bruk av reduce funksjonen, finner alle repetisjoner i en sang.

Algoritme:

1. Lag et HTML5 dokument.

2. Lag en *textarea* input boks med *id='inputId'*

3. Lag en knapp for å kjøre funksjonen

4. Lag en <p> output tekst boks med *id='output'*

5. Lag funksjonen *repetitions()*

    5.1 Definer ordet *lyrics* og sett *inputId* inn i den

    5.2 Definer *data* og ved bruk av *lyrics.split('\n')* del teksten inn i linjer

    5.3 Definer output og sett inn *data.reduce()* funksjonen i den

        5.3.1 lag en *if* funksjon som skal bruke *acc, currentValue, index, array* fra *reduce* funksjonen

            5.3.1.1 finn repetisjoner i linjene. *array.indexOf(currentValue)!=index && !acc.includes(currentValue)*

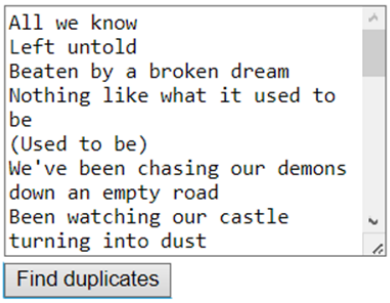
            5.3.1.2 Sett repetisjonene inn i *acc* ved bruk av *.push(currentValue)*

    5.4 Send outputen til <p> tekst boks ved hjelp av *document.getElementById("output").innerHTML*

```
<!DOCTYPE html>
<html lang="en">
<body>
```

```
<!-- Brukeren kan skrive inn teksten her -->
<textarea name="find repetitions" id="inputId" cols="30" rows="10"></textarea>
<br>
<!-- Brukeren kan kjøre funksjonen med knappen -->
<button onclick="repetitions()">Find duplicates</button>
<!-- Output vises her -->
<p id="output"></p>
<script>
//funksjonen for å finne repetisjoner
function repetitions(){
    //definerer lyrics og legger teksten inn i den
    let lyrics= document.getElementById("inputId").value;
    //definerer data og deler teksten inn i forskjellige strings
    let data = lyrics.split("\n");
    //definerer output og lager en reduce funksjon
    let output = data.reduce(
    //definerer acc, currentValue, index, og array
    function(acc,currentValue,index, array) {
    //finner repetisjonene i arrayen data
    if(array.indexOf(currentValue)!=index && !acc.includes(currentValue))
    //setter disse repetisjonene inn i acc
    acc.push(currentValue);
    //output blir til acc
    return acc;
    }, []);
    //sender outputen til output tekst boks <p>
    document.getElementById("output").innerHTML = output
}
</script>
</body>
</html>
```

Eksempel output:



Figur 36

5.3.11 Verb tense

Denne oppgaven kan endres for å passe andre språkfag også.

Oppgave:

Lag et program som kan finne ut hvilken verbtid som brukes i en tekst.

Oppgaven er hentet fra (Underwood et al., 2021)

Løsning:

Først skal vi lage en textarea boks for å skrive inn teksten. Så skal vi lage en knapp for å kjøre funksjonen, og en output tekst boks. Videre skal vi gjøre teksten om til en array, og lage en funksjon som, ved bruk av reduce funksjonen, finner alle ord som kan hjelpe med å bestemme hvilken verbtid som brukes. Etter det skal vi lage en funksjon som skal ta disse ordene og, ved hjelp av en if funksjon, bestemme hvilken verbtid som blir brukt.

Algoritme

1. Lag et HTML5 dokument.

2. Lag en textarea input boks med *id='inputId'*

3. Lag en knapp for å kjøre funksjonen

4. Lag en *<p>* output tekst boks med *id='output'*

5. Lag også en knapp for å kjøre den andre funksjonen

6. Lag en *<p>* output tekst boks med *id='output2'*

7. Lag funksjonen *identifiers()*

    7.1 Definer ordet *text* og sett *inputId* inn i den

    7.2 Definer *data* og ved bruk av *text.split(' ')* del teksten inn i ord

    7.3 Definer *output* og sett inn *data.reduce()* funksjonen i den

        7.3.1 lag en *if* funksjon som skal bruke *acc, currentValue* fra *reduce* funksjonen

            7.3.1.1 finn alle ord som ender med ed, og ing, og ord som was, is, will, have

            7.3.1.2 Sett ord inn i *acc* ved bruk av *.push(currentValue)*

    7.4 Send outputen til *<p>* tekst boks ved hjelp av *document.getElementById("output").innerHTML*

8. Lag funksjonen *identify()*

    8.1 definer *output2* for senere

    8.2 definer *words* og sett *output* fra forrige funksjonen inn i den

    8.3 lag en if funksjon

        8.3.1 bestem hvilken verbtid som passer til hver av de identifikatorer og sett svar inn i *output2*

    8.4 Send outputen til *<p>* tekst boks ved hjelp av *document.getElementById("output2").innerHTML*

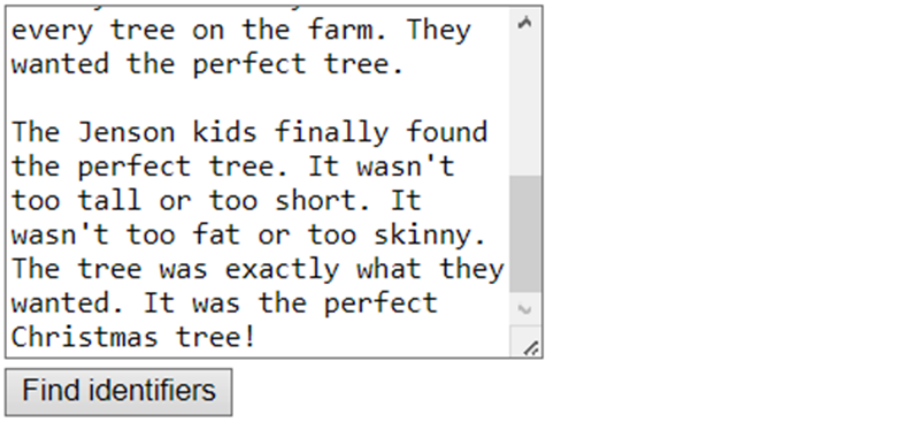
```
<!DOCTYPE html>
```

```
<html lang="en">
<body>
<!-- Brukeren kan skrive inn teksten her -->
<textarea name="find verb tense" id="inputId" cols="30" rows="10"></textarea>
<br>
<!-- Brukeren kan kjøre funksjonen med knappen -->
<button onclick="identifiers()">Find identifiers</button>
<!-- Output vises her -->
<p id="output"></p>
<br>
<!-- Brukeren kan kjøre funksjonen med knappen -->
<button onclick="identify()">Identify</button>
<!-- Output vises her -->
<p id="output2"></p>

<script>
//funksjonen for å finne identifikatorer
function identifiers(){
    //definerer text og setter teksten inn i den
    let text= document.getElementById("inputId").value;
    //definerer data og deler teksten inn i forskjellige ord strings
    let data = text.split(" ");
    //definerer output og lager en reduce funksjon
    let output = data.reduce(function(acc,currentValue) {
    //finner identifikatorene i arrayen data
    if(currentValue.endsWith("ed"))
    //setter disse identifikatorene inn i acc
    acc.push(currentValue);
    else if (currentValue.endsWith("ing"))
    acc.push(currentValue);
    else if(currentValue=="was")
    acc.push(currentValue);
    else if(currentValue =="is")
    acc.push(currentValue);
    else if(currentValue == "will")
    acc.push(currentValue);
    else if(currentValue == "have")
    acc.push(currentValue);
```

```
    //output blir til acc
    return acc;
    }, []);
    //sender outputen til output tekst boks <p>
    document.getElementById("output").innerHTML = output+","
}
//funksjonen for å bestemme hvilken verbtid disse identifikatorene er
function identify(){
    //definerer output2
    let output2;
    //definerer words og legger identifikatorene inn i den
    let words= document.getElementById("output").innerHTML;
    //bestemmer hvilken verbtid som passer til hver av de identifikatorer
    if (words.includes("will")){
        output2= "future tense"
    }
    else if (words.includes("is")){
        output2= "present tense"
    }
    else if (words.includes("was")){
        output2= "past tense"
    }
    else if (words.includes("have")){
        output2= "present tense"
    }
    else if (words.includes("ed,")){
        output2= "past tense"
    }
    else if (words.includes("ing,")){
        output2= "present tense"
    }
    else{
        output2= "likely present tense"
    }
    //sender outputen til output tekst boks <p>
    document.getElementById("output2").innerHTML = output2
}
</script>
```

```
</body>
</html>
```

Eksempel output:



Figur 37

6 Konklusjon

Oppgavebeskrivelsen som ble gitt av oppdragsgiver sa at målet var å finne ut hvilke oppgaver innen språk i skolen som egnet seg best til programmering for å øke dybdeforståelse, engasjement og kreativitet hos elever. Man skulle også bestemme seg for hvilke fremgangsmåter for implementasjon av disse som passer.

Resultatet av utforskning til oppgaven viser at det er viktig at løsningen sørger for at:

- Oppgavene vil være interessante, utfordrende og engasjerende for elevene slik at de vil bli motiverte til å finne løsningen til det gitte problemet av eget initiativ
- Oppgavene vil måtte velges og være satt opp slik at den tvinger eleven til å måtte tenke algoritmisk, finne mønster og være løsningsorientert

- Oppgavene følger pensum og kompetansemål, og har en tydelig framgangsmåte slik at lærere enkelt vil kunne sette seg inn i programmeringen og kunne lære elevene sine effektivt
- Oppgavene i språkfag kan anvendes til programmering slik at elevene vil kunne lære språkfagets pensum og samtidig få en dypere forståelse for koding og digitale verktøy

I dette prosjektet evalueres påliteligheten til oppgavene ved å se om de følger pensum og kompetansemålene til språkfagene. Dette er blitt gjort ved å plukke ut oppgaver fra språkfagenes fagbøker som kan anvendes til programmering og dermed anvende dem.

Dokumentet i seg selv må også evalueres ved å teste oppgavene og algoritmene deres med noen språkfaglærere som testbrukere. Det vil testes om de ender opp med å klare oppgavene med samme framgangsmåte som beskrevet i dokumentet eller om en annen metode er mer effektiv. Denne evalueringsdelen vil ikke bli gjennomført i dette prosjektet på grunn av mangel på tid, men kan gjøres i videre arbeid.

Ellers er det mulighet for forbedring av dokumentet ved at oppgaver fra eventuelle nye fagbøker ved endring av pensum/kompetansemål kan inkluderes/erstatte udaterte oppgaver. Man kan også inkludere oppgaver som man mener er enda mer relevante for pensum og kan utfordre elevene mer.

Referanser

American Historical Association (1884) Historical Thinking Skills. Tilgjengelig fra: https://www.historians.org/teaching-and-learning/teaching-resources-for-historians/teaching-and-learning-in-the-digital-age/the-history-of-the-americas/the-conquest-of-mexico/for-teachers/setting-up-the-project/historical-thinking-skills (Hentet: 16.04.2023).

Balanskat, A., & Engelhardt, K. (2015). *Computing our future: Computer programming and coding—Priorities, school curricula and initiatives across Europe.*

http://www.eun.org/documents/411753/817341/Computing+our+future_final_2015.pdf/d3780a64-1081-4488-8549-6033200e3c03

Blichfeldt, K., Heggen, T.G. og Huseby, Å. (2020) *Kontekst 8-10.* 3. utg, 5. opplag. Gyldendal Norsk Forlag

Brocke, J. vom, Hevner, A., & Maedche, A. (2020). Introduction to Design Science Research. I *Design Science Research. Cases* (ss. 1–13).

Burgess, R., & Sørhus, T. B. (2013). *Access to English* (1. utgave). Cappelen Damm.

Futschek, G. (2006). Algorithmic Thinking: The Key for Understanding Computer Science. I R. T. Mittermeir (Red.), Informatics Education – The Bridge between Using anf Understanding Computers (s. 159-168. Springer, Berlin, Heidelberg. https://link.springer.com/chapter/10.1007/11915355_15

Kunnskapsdepartementet. (2004). *St.meld. Nr. 30(2003-2004): Kultur for læring.* https://www.regjeringen.no/contentassets/988cdb018ac24eb0a0cf95943e6cdb61/no/pdfs/stm200320040030000dddpdfs.pdf

Kunnskapsdepartementet. (2006). *Kunnskapsløftet: Reformen i grunnskole og videregående opplæring.* https://www.regjeringen.no/globalassets/upload/kilde/ufd/prm/2005/0081/ddd/pdfv/256458-kunnskap_bokmaal_low.pdf

Kunnskapsdepartementet. (2007). *Kunnskapsløftet presentasjon.* https://www.regjeringen.no/globalassets/upload/kd/vedlegg/kunnskapsloeftet/kunnskapsloftet_presentasjon.pdf

Kunnskapsdepartementet. (2016). *Meld. St. 28 (2015–2016): Fag – Fordypning – Forståelse—En fornyelse av Kunnskapsløftet.* https://www.regjeringen.no/contentassets/e8e1f41732ca4a64b003fca213ae663b/no/pdfs/stm201520160028000dddpdfs.pdf

Kunnskapsdepartementet. (2018). *Fornyer innholdet i skolen.* https://www.regjeringen.no/no/dokumentarkiv/regjeringen-solberg/aktuelt-regjeringen-

solberg/kd/pressemeldinger/2018/fornyer-innholdet-i-skolen/id2606028/?expand=factbox2617567

Ludvigsenutvalget. (2014). *Elevenes læring i fremtidens skole—Et kunnskapsgrunnlag.* https://www.regjeringen.no/contentassets/e22a715fa374474581a8c58288edc161/no/pdfs/nou201420140007000dddpdfs.pdf

Ludvigsenutvalget. (2015). *Fremtidens skole—Fornyelse av fag og kompetanser.* https://www.regjeringen.no/contentassets/da148fec8c4a4ab88daa8b677a700292/no/pdfs/nou201520150008000dddpdfs.pdf

Nätt, T. H. (2023). *Lavnivåspråk* https://snl.no/lavniv%C3%A5spr%C3%A5k

Pettersen, R. C. (2017). *Problembasert læring for studenter og lærere. Introduksjon til PBL og studentaktive læringsformer 3. Utg.*

Pettersen, S., & Røkaas, F. (2020). *Stages 8. Engelsk for ungdomstrinnet.* (2. utgave). Aschehoug undervisning.

Pettersen, S., & Røkaas, F. (2021). Stages 9. *Engelsk for ungdomstrinnet.* (2. utgave). Aschehoug undervisning.

Rossen, E. (2022). *Programmering (IT)* https://snl.no/programmering_-_IT

Sawyer, R. K. (Red.). (2006). The New Science of Learning. I *The Cambridge Handbook of the Learning Sciences.*

Sevik, K. (2016). *Programmering i skolen*. https://www.udir.no/globalassets/filer/programmering_i_skolen.pdf

Statistics Canada. (2020). *2006 Census Topic-based tabulations.* https://census.gc.ca/census-recensement/2006/dp-pd/tbt/Rp-eng.cfm?LANG=E&APATH=3&DETAIL=1&DIM=0&FL=A&FREE=1&GC=0&GID=0&GK=0&GRP=1&PID=92335&PRID=0&PTYPE=88971&S=0&SHOWALL=No&SUB=0&Temporal=2006&THEME=80&VID=0&VNAMEE=&VNAMEF=

TechnoKids (2018) *5 New Features in Scratch* . Tilgjengelig fra: https://www.technokids.com/blog/programming/scratch/5-new-features-in-scratch-3-0/ (Hentet: 23.04.2023).

Underwood, I., Birkeland, A., Pettersen, S. A., & Hunstadbråten, S. (2021). *Edge—Programfaget engelsk 1, vg2/3—Studieforberedende utdanningsprogram.* Gyldendal.

Universitetet i Oslo. (2021). *5E-modellen.* https://www.uv.uio.no/forskning/satsinger/fiks/kunnskapsbase/elevaktive-arbeidsformer/Metoder%20og%20modeller/5e-modellen/

Universitetet i Oslo. (2021). *Prosjektarbeidsmetoden.* https://www.uv.uio.no/forskning/satsinger/fiks/kunnskapsbase/elevaktive-arbeidsformer/Metoder%20og%20modeller/prosjektarbeidsmetoden/

Universitetet i Oslo. (2021b). *Storyline.* https://www.uv.uio.no/forskning/satsinger/fiks/kunnskapsbase/elevaktive-arbeidsformer/Metoder%20og%20modeller/storyline/

Universitetet i Oslo. (2022). *Digital dekning i Norges 100 største kommuner*. https://www.uv.uio.no/forskning/satsinger/fiks/kunnskapsbase/digitalisering-i-skolen%20%28tidligere%20versjon%29/digital-dekning-i-norges-100-storste-kommuner/

Universitetet i Oslo. (2023). *Metoder og modeller.* https://www.uv.uio.no/forskning/satsinger/fiks/kunnskapsbase/elevaktive-arbeidsformer/Metoder%20og%20modeller/

Utdanningsdirektoratet. (2019a). *Hva er nytt i engelsk fordypning?* https://www.udir.no/laring-og-trivsel/lareplanverket/fagspesifikk-stotte/nytt-i-fagene/hva-er-nytt-i-engelsk-fordypning/

Utdanningsdirektoratet. (2019b). *Hva er nytt i engelsk og engelsk for elever med tegnspråk?* https://www.udir.no/laring-og-trivsel/lareplanverket/fagspesifikk-stotte/nytt-i-fagene/hva-er-nytt-i-engelsk-og-engelsk-for-elever-med-tegnsprak/

Utdanningsdirektoratet. (2019c). *Hva er nytt i fremmedspråk?* https://www.udir.no/laring-og-trivsel/lareplanverket/fagspesifikk-stotte/nytt-i-fagene/hva-er-nytt-i-fremmedsprak/

Utdanningsdirektoratet. (2019d). *Hva er nytt i norsk?* https://www.udir.no/laring-og-trivsel/lareplanverket/fagspesifikk-stotte/nytt-i-fagene/hva-er-nytt-i-norsk/

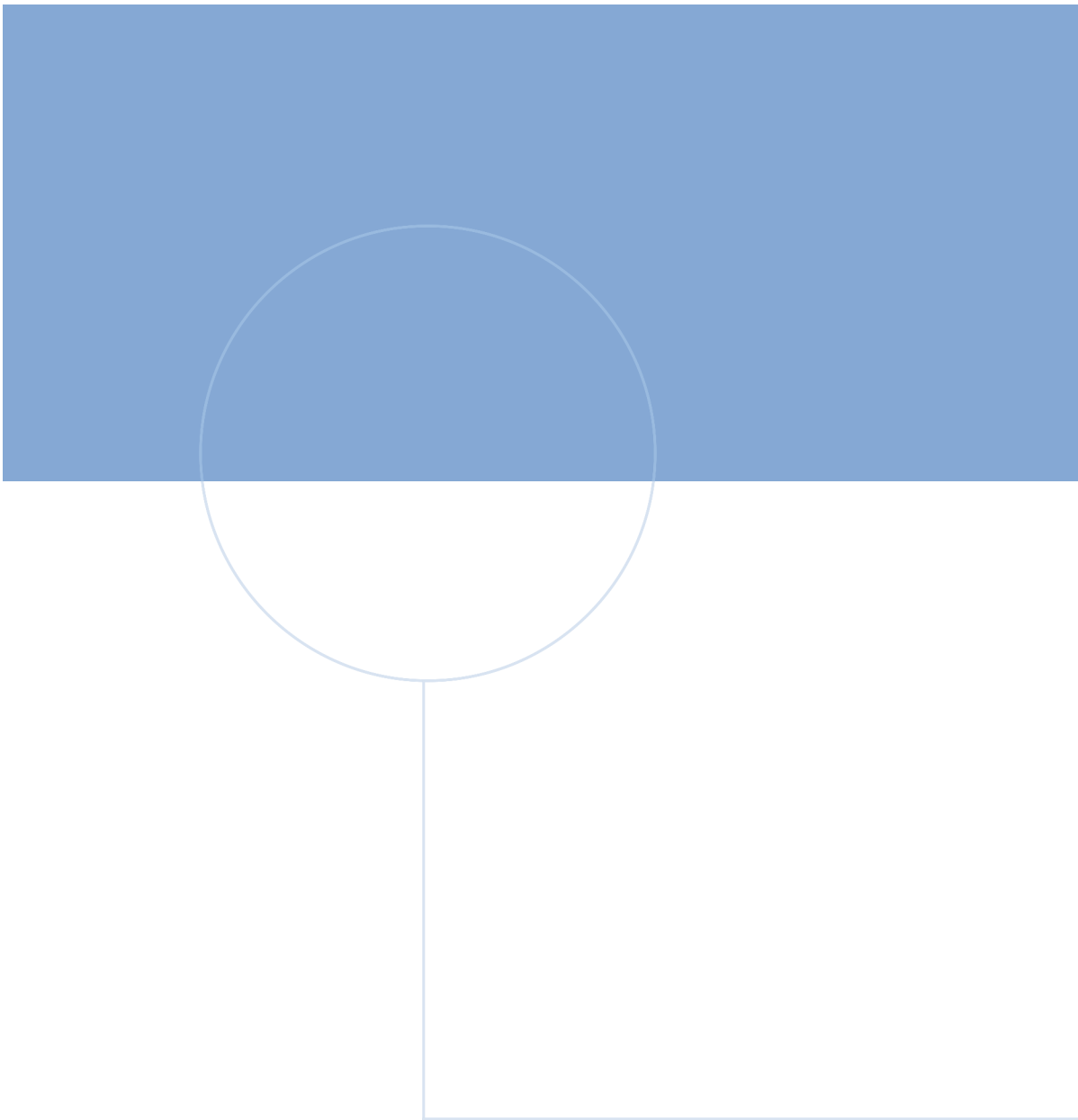
Utdanningsdirektoratet. (2019e). *Algoritmisk Tenkning* https://www.udir.no/kvalitet-og-kompetanse/profesjonsfaglig-digital-kompetanse/algoritmisk-tenkning/

Utdanningsdirektoratet. (2020). *Utdanningsspeilet 2020*. https://www.udir.no/tall-og-forskning/publikasjoner/utdanningsspeilet/utdanningsspeilet-2020/del-2/digital-undervisning-under-koronastengte-skoler/

Utdanningsdirektoratet. (2021). *Hvorfor har vi fått nye læreplaner?* https://www.udir.no/laring-og-trivsel/lareplanverket/stotte/hvorfor-nye-lareplaner/

Visual Studio Code (2015) *Emmet in Visual Studio Code* https://code.visualstudio.com/docs/editor/emmet (Hentet: 23.04.2023)

Waite, J., & Sentance, S. (2021). *Teaching programming in schools: A review of approaches and strategies*. https://www.raspberrypi.org/app/uploads/2021/11/Teaching-programming-in-schools-pedagogy-review-Raspberry-Pi-Foundation.pdf



NTNU

Kunnskap for en bedre verden