Utdannings-
direktoratet

Veiledning til programmering valgfag

ARTIKKEL | SIST ENDRET: 30.06.2016

Innhold

Innledning

Denne veiledningen gir råd og tips om hvordan du kan arbeide med læreplanen i valgfaget. Veiledningen er ikke ment å være heldekkende ved at eksemplene dekker alle kompetansemålene i læreplanen, men den gir

noen tips og råd til hvordan lærere kan jobbe praktisk med det faglige innholdet i valgfaget.

Veiledningen er ikke et formelt dokument som kan erstatte eller sidestilles med selve læreplanen i valgfaget. Utdanningsdirektoratet er ansvarlig for innholdet i veiledningen, og den er utarbeidet i samarbeid med relevante fagmiljøer og fagpersoner.

Fagets egenart

Å gi elever grunnleggende ferdigheter i programmering er med på å forberede dem til fremtidig arbeidsliv og samfunnsliv. Nesten alle områder i samfunnet preges av økt automatisering, for eksempel innen helse, fiskeri, transport og varehandel. Mange av fremtidens jobber vil kreve forståelse for hvordan digital teknologi fungerer. Kunnskap om programmering er viktig for å forstå muligheter og begrensninger ved teknologien som omgir oss. Programmering innebærer å skape digitale produkter, ikke bare å bruke dem. Denne kompetansen kan nyttiggjøres innen en rekke fagområder, og slik bidra til å styrke og utvide mulighetene i andre fag på skolen.

Elevene skal gjennom valgfaget lære å programmere. Elever som velger faget, skal designe og utvikle egne dataprogrammer, både individuelt og sammen med andre. De skal også forbedre eksisterende produkter. Valgfaget legger til grunn en bred forståelse av fagområdet programmering. I dette inngår prosessen fra å identifisere problemer og utforme mulige løsninger, til å lage kode som kan forstås av en datamaskin, systematisk feilsøke og forbedre denne koden, og dokumentere løsningen på en forståelig måte. Dette fører til at elevene opparbeider en forståelse av hvordan en datamaskin fungerer, og hvilke bruksområder den kan ha.

Faget bygger på elementer fra matematikk og naturvitenskapelige fag, og det åpner for å utforske verden gjennom simuleringer. Elevene kan også erfare nytten av datamaskiner ved å lage programmer som gjør utregninger knyttet til realfagene. Når flere utregninger skal gjøres, får de erfare hvordan programmer fungerer, samt hvor effektivt et program kan være. I tillegg styrker programmering sentrale ferdigheter i realfagene, f.eks. oppbygging av logiske resonnementer og feilsøking.

Hovedområdene modellering og koding

Valgfaget er delt inn i to hovedområder: modellering og koding. Hovedområdene må ses i sammenheng. Modellering og koding er ikke adskilte aktiviteter, og eleven vil som oftest veksle mellom disse hovedområdene når de utvikler programmer.

Modellering
Modellering handler i programmering om prosessen med å komme fram til en overordnet beskrivelse eller modell av det som skal utføres av dataprogrammet. Til dette hører kunnskap om hvordan datamaskiner virker, og hva slags problemer som er egnet for å løse ved programmering. Elevene vil gjennom arbeidet med modellering få mulighet til å utvikle algoritmisk tankegang og arbeide med problemløsning.

Modellering av matematiske og naturvitenskapelige fenomener er også en sentral del av hovedområdet. Modellering i denne sammenheng handler om å lage en forenklet framstilling (en modell) av virkeligheten og representere den ved hjelp av en datamaskin.

Koding
Koding er prosessen med å skrive og utvikle programmer ved hjelp av programmeringsspråk, og dette krever i stor grad at elevene bruker en datamaskin eller andre programmerbare enheter. Hovedområdet koding er nært knyttet til selve aktiviteten å skrive programkode i ulike programmeringsspråk. Elevene skal anvende vanlige prinsipper for utforming av programmer og forstå hensikten med dem.

Arbeidsmåter i programmering

Det er et mål at elevene kjenner til og bruker god programmeringsskikk, slik som å kommentere og dokumentere eget arbeid underveis. God programmeringsskikk inkluderer også vanlige måter å organisere arbeidet med programutvikling på i næringslivet. Punktene nedenfor beskriver noen slike arbeidsmåter i programmering.

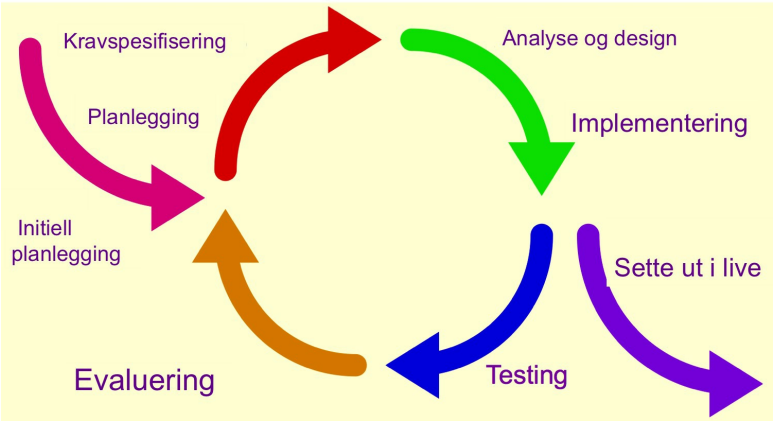
Ulike tilnærminger til programutvikling

Det er vanlig å veksle mellom to hovedtilnærminger til utvikling av programkode: ovenfra-og-ned og nedenfra-og-opp. Denne veiledningen refererer ofte til ovenfra-og-ned ved at vi legger vekt på å begynne med en overordnet problemstilling, for så å bryte denne ned i mindre delproblemer som kan løses ved hjelp av programmering. For eksempel skal elevene lage et ping-pong-spill. De kan begynne med å beskrive hvordan et slikt spill virker, hvilke funksjoner spillet skal ha (ball som spretter, fart, vinkler, osv.), og deretter programmere løsninger for de ulike delene av hovedproblemet (hvordan lage ping-pong-spill).

I praksis vil elevene like ofte arbeide nedenfra-og-opp ved at de starter med å utvikle små programmer som løser mindre del-problemer, og siden bygger disse sammen i enheter som løser mer komplekse problemer. Et eksempel er å gå fra å lage et program som simulerer en ball som spretter, til å utvikle et ping-pong spill.

Iterativ og inkrementell prosess

Utvikling av dataprogram innebærer som regel å repetere en prosess eller deler av prosessen for å komme stadig nærmere et ønsket mål eller et resultat. Hver repetisjon kaller vi en iterasjon, og for hver iterasjon legger eleven til noe (mer funksjonalitet, forbedret design, osv.). Det kaller vi en inkrementell prosess. Det innebærer at programmereren ofte tester ut deler av programmet og gjør justeringer i modellen og designet fortløpende.



De ulike fasene i en utviklingsprosess for å lage et dataprogram.

Figuren viser ulike faser av en programutviklingsprosess. Først må vi se på hva vi ønsker å lage, og utarbeide en kravspesifikasjon ut ifra det. Her det det viktig å ta hensyn til hva programmet skal gjøre, og hvem som skal bruke det. Så må programmet planlegges før vi kan sette i gang med implementering. Da er det viktig å tenke på hvordan programmet senere skal kunne vedlikeholdes eller videreutvikles, når vi bestemmer strukturen. Når programmet er implementert, må vi teste om programmet fungerer og gjør alt det skal. Ofte vil vi erfare at noen ting fungerer og noen ting fungerer ikke, og da må vi evaluere hvordan vi kan forbedre og ferdigstille det som gjenstår. Når det er gjort, må vi implementere de nye spesifikasjonene og rette feil før vi så tester igjen. Denne prosessen gjentas frem til programmet fungerer slik det skal, og så kan settes ut i livet.

Valg av programmeringsspråk

Det finnes mange forskjellige programmeringsspråk. Alle har sine fordeler og ulemper, og det er viktig å velge et programmeringsspråk som egner seg til oppgaven eleven skal utføre. Vi deler dem ofte inn i blokkbaserte eller tekstbaserte programmeringsspråk, ut ifra hvordan kommandoene er representert. Blokkbaserte programmeringsspråk benytter seg av klosser som eleven setter sammen til større blokker og programmer, litt som å bygge med LEGO. Tekstbaserte programmeringsspråk består av tekstbaserte kommandoer, og det er viktig å følge korrekt syntaks for å få programmet til å kjøre.

Læreplanen legger ingen føringer for hvilke programmeringsspråk eller verktøy for programmering som skal benyttes i undervisningen i faget, men det er et krav at elevene i løpet av skoleåret bruker minst to ulike programmeringsspråk i sitt arbeid. Det er spesifisert i læreplanen at minst ett av språkene skal være tekstbasert.

Ulike språk har ulike styrker og bruksområder, for eksempel er språket “JavaScript” mye brukt når man lager ting på nett, mens Lua er et språk som er mye brukt i spill-utvikling. I valg av programmeringsspråk som skal benyttes i valgfaget, bør du som lærer vurdere følgende:

- Hva slags erfaring og kompetanse du selv har.
- Hva slags kompetanse som er tilgjengelig i nærmiljøet (kollegaer på egen eller nærliggende skoler, ev. på videregående skoler, høgskoler og universitet i nærheten, IT-bedrifter du kan samarbeide med, osv.).

- Hva slags oppgaver du tenker å arbeide med i faget (nettutvikling, roboter, dataspill, osv.).
- Hvor vanskelig eller lett det er å komme i gang med de ulike språkene.

Dersom læreren (lærerne) i faget har begrenset programmeringserfaring, kan det være lurt å starte med blokkprogrammering i Scratch og gå videre til tekstbasert programmering i Python eller JavaScript. Men det kan være gode grunner til å velge noe annet, for eksempel hvis skolen har investert i programmerbare roboter som Lego Mindstorms og Spheros eller mikrokontrollere som Arduino. Elevenes erfaring og kompetanse vil også påvirke valg av programmeringsspråk, for eksempel kan de bruke mer avanserte, tekstbaserte språk etter hvert.

Elever og valg av programmeringsspråk

Etter hvert som elevene får erfaring med flere ulike programmeringsspråk, vil det å velge rett verktøy (språk) til oppgaven de skal løse, også være en viktig oppgave. Elevene kan gjerne bli bedt om å vurdere ulike språk når de skal løse en oppgave, og hvordan de argumenterer for sitt valg, kan være del av vurderingsgrunnlaget.

Samarbeid og delingskultur

Programutvikling gjøres sjelden av enkeltpersoner alene, blant annet fordi oppgavene kan være store og komplekse. Det er en utbredt kultur for å dele sitt arbeid med andre og å bygge videre på andres programkode. I industrien gjøres programvareutvikling ofte av et team av programmerere som jobber sammen på et prosjekt. Dette bør reflekteres i undervisningen i valgfaget. Flere av de praktiske eksemplene i kapittel 3 legger til rette for at valgfagsklassen deles i samarbeidsgrupper på 2-5 personer. Det er flere ting det er lurt å ta hensyn til i gruppeinndelingen. For eksempel kan faglig homogene grupper være både inspirerende og gi faglig utfordringer til elever på alle nivåer. I aldersblanda klasser kommer dette virkelig til sin rett. I tillegg kan det være lettere å gi tilpasset undervisning til alle elever, uavhengig av om de har hatt valgfaget tidligere eller ikke.

Det er nyttig å skaffe seg en oversikt tidlig over hvilke elever som har programmert tidligere, enten i andre skolefag, via Kodeklubben eller på egen hånd, og hvilke kompetanser disse elevene har. I tillegg vil noen elever ha faget for andre og tredje år.

Samarbeidsformer og delingskultur

En arbeidsmetode som er mye brukt i såkalt smidig utvikling kalles ‘pair programming’ der to programmerere sitter sammen på samme arbeidsstasjon for å skape kode. Paret har hver sin rolle som byttes hyppig: ‘sjåføren’ skriver kode mens ‘kartleseren’ holder et mer overordnet blikk og kontrollerer at paret beveger seg i riktig retning.

Andre samarbeidselementer kan være å tipse hverandre om nyttige ressurser knyttet til det temaet elevene jobber med. Læreren kan presentere en grunnpakke med ressurser, og i tillegg legge til rette for at elevene kan bidra med tips til nettsteder og ressurser som kan brukes til oppgavene. Tipsene kan for eksempel deles med klassen gjennom et samskrivingsdokument, eller som innlegg i et forum på fagsiden på skolens

læringsplattform. Det finnes en rekke gode ressurser knyttet til programmering, og siden dette er et fag i utvikling, vil det også stadig være nye ressurser som dukker opp på markedet.

Samarbeid i programmering kan også være å bygge videre på hverandres eller andres kode. Det kan innebære å gi tilbakemelding på hverandres kode og programmer. For eksempel kan elevene dele spillene sine med de andre i klassen, etterfulgt av at andre elever spiller det, og gir tilbakemelding på sin spillopplevelse og setter ord på hva som var bra, og hvilke mulige forbedringer som kan gjøres.

Det er flere måter å stimulere en delingskultur i klasserommet på. For eksempel kan samskriving, inkludert kommentering og tilbakemelding på hverandres arbeid, være en arbeidsmåte. En annen måte er å få elevene til å sette ord på det de har lært gjennom små minipresentasjoner for gruppen eller i klassen. En slik minipresentasjon kan være å vise hvilken tag eleven bruker for å lage linjeskift i HTML, eller hvordan få Scratch til å gå ti skritt bakover. Det er vesentlig at elevene viser koden sin, forklarer den, samt viser hvordan det vil se ut når programmet kjører. Gjennom en slik deling vil elevene få inspirasjon av hverandres arbeid, erfare ulike løsningsmetoder gjennom praktiske eksempler og dele tips og triks. Minipresentasjonen kan være et fint utgangspunkt for en gruppesamtale rundt alternative løsningsmetoder eller om overføringsverdien av eget arbeid.

Det finnes mange delingsarenaer for programkode, og elevene kan gjerne lære å finne eksisterende programmer og dele sine egne programmer på slike steder. For eksempel kan eleven i Scratch dele sine produkter med andre brukere rundt omkring i verden og “remikse” hverandres programmer. Et annet eksempel er GitHub, et nettsted der programutviklere deler programkode og foreslår endringer i eksisterende programmer. Det kan være motiverende å få tilbakemelding på det du har laget fra andre ‘kodere’, enn fra dem du går på skole med.

Algoritmisk tankegang

Det er et mål for faget at elevene utvikler algoritmiske tankegang og evne til å løse problemer systematisk. Begrepet viser til det engelske “computational thinking skills” som kan forklares som “å tenke som en informatiker”. Algoritmisk tankegang innebærer at elevene lærer å tilnærme seg problemer på en systematisk måte, og foreslår løsninger som de kan bruke datamaskiner til å løse helt eller delvis. Det vil si både

å tenke på hvilke steg som skal til for å løse et problem

å bruke sin teknologiske kompetanse for å få datamaskinen til å løse problemet

Algoritmisk tankegang innebærer å bryte ned store, komplekse problemer til mindre, mer håndterlige del-problemer. Det er en problemløsningsprosess som inkluderer å organisere og analysere data på en logisk måte og å lage fremgangsmåter for å løse komplekse problemer. Det handler også om å lage modeller av den virkelige verden og å generalisere løsninger slik at den kan anvendes til å løse lignende problemer. Denne måten å arbeide på er sentral i programvareutvikling, men kan også brukes som metode i mange andre sammenhenger og fag. Dette beskrives nærmere i eksempelet “Fra problem til delproblem”.

Praktiske eksempler

Her vil du finne tips og råd ved planlegging, gjennomføring og vurdering av arbeidet med valgfaget. Det faglige innholdet i opplæringen vil variere ut fra lokalt arbeid med læreplanene. Du vil også finne en idébank som kan inspirere til egen planlegging.

Variasjon i innhold blir tatt opp med utgangspunkt i rammene for læreplanene knyttet til kompetansemål og vurdering. Videre følger det praktiske eksempler på hvordan lærere kan jobbe med innholdet ut fra hovedområdene og kompetansemålene i valgfaget.

Enkelte av eksemplene viser kompetansemål som er konkretisert i læringsmål, og som viser eksempler på kjennetegn på høy måloppnåelse. Dette kan brukes som grunnlag i underveisvurdering, slik at det er kjent for elevene hvor de er i opplæringen, og hva som er grunnlaget for videre læring.

Læreplanene i valgfag og variasjon i innhold

Læreplanene i valgfag skiller seg ut fra de andre læreplanene i Kunnskapsløftet ved at valgfagene har kompetansemål som beskriver ett års opplæring, men de skal kunne brukes på alle tre trinn i opplæringen. Elever har også mulighet til å velge samme valgfag om igjen. Dette betyr at alle elever, uansett om de har faget i ett år eller over flere år, skal ha opplæring i - og vurderes ut fra de samme kompetansemålene. Opplæringen i faget skal avsluttes hvert år.

Intensjonen med valgfagene er blant annet å motivere elevene. Hvordan kan elever som har hatt valgfaget tidligere, oppleve variasjon i valgfaget?

Valgfaggruppen kan være mer sammensatt enn i andre fag, for eksempel ved aldersblanding av elever eller ved at noen elever har hatt valgfaget tidligere, mens andre har valgfaget for første gang. Det er viktig å ta hensyn til sammensetningen av gruppen når det gjelder variasjon i valg av innhold, osv. Det er ikke trinnet eleven går på som skal styre innholdet, og det skal heller ikke vektlegges at elever har hatt valgfaget tidligere. Ettersom valgfagene har kompetansemål som skal oppnås etter ett år, vil det likevel være hensiktsmessig at elever som velger samme valgfag om igjen, opplever variasjon i faget fra ett år til det neste.

Å gjennomføre en opplæring i valgfag med progresjon og variasjon løses, som i alle andre fag, innenfor tilpasset opplæring. Variasjon og progresjon kan for eksempel være forskjell i innhold, arbeidsoppgaver, aktiviteter, lærestoff, intensitet (tempo) og organisering av opplæringen, arbeidsmåter, læringsarenaer, m.m. Utgangspunktet for opplæringen skal være at de samme kompetansemålene gjelder for alle elevene, og at de gir rom for å velge variert innhold ut fra lokale forhold og elevenes forutsetninger.

Eksempler på ulike muligheter for variasjon

- Tilpasse valgmuligheter i innhold og tema innenfor de samme kompetansemål til ulike elever.
- Det samme valgfaget kan ha variasjon knyttet til ulike aktiviteter og innhold fra ett år til det neste.

Dette kan sikre at elevene opplever og lærer noe nytt om de velger samme faget over flere år.

- Elever som ønsker større utfordringer, kan tilbys mer komplekse læringsaktiviteter innenfor de samme kompetansemålene i læreplanen. Dette kan være en måte å tilpasse innholdet i opplæringen på, sikre variasjon og motivasjon, selv om kompetansen er den samme for alle elevene.
- Valgfagene kan være prosjektbaserte. Dette åpner opp for variasjon og samarbeidsoppgaver mellom ulike elever fra ulike trinn og mellom ulike valgfag.

Variasjon i programmeringsfaget

Valgfaget programmering egner seg godt til variasjon i innhold og tema.

Variasjon i innhold
kan være å bruke ulike programmeringsspråk og verktøy til å løse oppgavene. Enten ved at hele elevgruppen bruker samme verktøy og språk, eller ved at undervisningen baserer seg på at disse er ulike fra år til år, eller ved at elever på ulikt nivå i elevgruppen bruker ulike verktøy til å løse oppdraget. De aller fleste læringsaktivitetene i faget kan tilpasses slik at de både kan løses forholdsvis enkelt (for eksempel ved å lage en animasjon i et visuelt programmeringsverktøy) eller svært avansert (for eksempel å utvikle en komplett, fungerende løsning i et tekstbasert språk eller en robot som fungerer).

Variasjon i tema
kan være å arbeide med et større prosjekt hvert år, for eksempel Mars-landing det ene året og robotstøvsuger det neste. Variasjon i tema kan også være å periodisere undervisningen slik at ulike tema undervises i ulike semestre. Eksempel på slike tema kan være: nett-utvikling, spill og spilldesign, robotprogrammering, styring av mikrokontrollere og utvikling av apper for mobil. På denne måten vil både elever som tar faget første gang, og elever som har tatt faget tidligere år, oppleve innholdet som nytt.

Programmering uten datamaskin

Oppgaven egner seg som en oppstartsaktivitet i den første undervisningstimen i faget. Dette er en praktisk øvelse for å hjelpe elevene til å forstå hvordan datamaskiner og programmer fungerer på et overordnet nivå. Elevene skal lære å sette sammen en serie av instruksjoner til en algoritme som løser en gitt oppgave.

Kompetansemål

- Gjøre rede for hvordan datamaskiner og programmer fungerer, inkludert et utvalg utbredte programmeringsspråk og deres bruksområder
- Bruke grunnleggende prinsipper i programmering, slik som løkker, tester, variabler, funksjoner og enkel brukerinteraksjon

Læringsmål

- Elevene kan forklare hva som er forskjellen på en datamaskin og et program
- Elevene kan forklare hva algoritmer og løkker er

Utstyr

- Et rutenett på gulvet, for eksempel av teip
- Lapper til å skrive instruksjoner på

Oppgave til elevene

Programmering handler om å fortelle datamaskinen hva den skal gjøre. Læreren kan gjerne starte med en diskusjon om hvor smart en datamaskin er. Datamaskiner gjør det den får beskjed om, og i den rekkefølgen beskjedene gis. Det er derfor viktig å være nøyaktig når vi gir instruksjoner til datamaskinen. Flere instruksjoner etter hverandre utgjør en algoritme.

Elever programmerer hverandre

- Lag en labyrint i klasserommet, for eksempel ved å tegne et rutenett på gulvet. Målet er å finne raskeste vei fra start og til et gitt punkt.
- Elevene planlegger instruksjonene ved å bruke piler, symboler eller kommandoer som "gå tre ruter fram, sving til høyre, gå to fram, osv."
- Når programmet er ferdig, kan det gis til en annen elev, som så skal følge instruksene gjennom labyrinten.
- Elevene kan bytte på å være programmere, det vil si de som gir instruksjoner, og roboter, de som blir programmert.
- Havnet roboten der den skulle? Hvis ikke, er det viktig å feilsøke og finne hvor feilen ligger.

Læreren kan introdusere elevene for begreper som algoritmer og løkker. Ved bruk av løkker ('gjenta X ganger') kan elevene finne fram til færrest antall instruksjoner som leder til målet. Dette kan også gjøres ved å bruke spillbrett av typen stigespillet: Hva er minste antall instruksjoner som behøves for å komme i mål?

Tilpasning til ulike elevgrupper

Oppgaven er inkluderende for de aller fleste elever. Liknende aktiviteter har vært gjennomført helt ned på barnehagenivå, samtidig som man kan øke vanskegraden og gi elevene mer kompliserte oppgaver. Andre eksempler finner du på csunplugged.org. og Robotvenner fra kodeklubben.

Elever som har programmert en del før, kan likevel ha nytte av å konkretisere innholdet i faget ved å bruke sine og andres kropper. Alternativt kan aktivitetene også gjennomføres på datamaskin, for eksempel ved å lage labyrinter eller spill i Scratch. Elever som har hatt faget før, kan også få oppgave i å lage et opplegg med analog programmering for medelever som er nybegynnere.

Vurdering av elevene

Dette er en introduksjonsaktivitet som ikke nødvendigvis egner seg for vurdering. Men den kan vurderes ut fra hvor godt elevene får til å gi instruksjoner, og hvor godt de kan forklare hva aktiviteten har med algoritmer og løkker å gjøre.

Kjør en kodetime med elevene

I starten kan det være lurt å gi elevene en kort introduksjon til viktige begreper og konsepter de vil møte på når de programmerer. Det finnes mange slike opplegg som er tilpasset nybegynnere, og som det tar en skoletime å gjennomføre.

Kompetansemål

- Gjøre rede for hvordan datamaskiner og programmer fungerer, inkludert et utvalg utbredte programmeringsspråk og deres bruksområder
- Bruke grunnleggende prinsipper i programmering, slik som løkker, tester, variabler, funksjoner og enkel brukerinteraksjon

Læringsmål

- Elevene kan forklare hvordan et dataprogram fungerer
- Elevene kan forklare hva algoritmer, løkker og tester er, og hvorfor vi bruker det i programmering

Utstyr

- Datamaskin eller nettbrett med internett
- Nettstedet code.org
- Nettstedet kodeklubben.no (oppgaver om Scratch)

Fremgangsmåte

På nettstedet Code.org finnes en rekke opplegg for å gjennomføre en kodetime der elevene skal styre diverse figurer kjent fra spill og filmer ved hjelp av kode. Underveis vil elevene lære å bruke grunnleggende konsepter i programmering. Undervisningsoppleggene for kodetimen er oversatt til bokmål og nynorsk, og nye opplegg publiseres jevnlig. Elevene kan for eksempel jobbe seg gjennom et av følgende opplegg:

- Frost
- Star Wars
- Minecraft

I disse oppleggene introduseres algoritmer, løkker og tester ved hjelp av blokk-programmering, og de legger et grunnlag for å gå videre med programmering i andre språk. Dette er opplegg som kjøres i nettleseren, og som fungerer fint både på datamaskiner og nettbrett. Understrek at elevene må se videoene som dukker opp underveis. Her forklares mange begreper og konsepter i programmering på en enkel måte av 'IT-kjendiser' som Mark Zuckerberg og Bill Gates. La gjerne elevene forklare med egne ord hvorfor vi bruker løkker, hva en algoritme er, osv.

Tilpasning til ulike elevgrupper

Aktivitetene på code.org finnes også på en rekke ulike språk, noe som kan være en god hjelp for fremmedspråklige elever. Enkelte av oppleggene kan utføres både med blokker av naturlig språk og blokker av programmeringsspråk, eller ved at elevene skriver programmeringsspråk. Code.org har også et lærerverktøy som lar deg opprette en klasse med elevene dine, slik at du kan se progresjonen for den enkelte elev. Det kan være et godt utgangspunkt for å veilede og differensiere. For elever som trenger litt større utfordringer, kan kodeklubbens Kodetime-oppgaver i Scratch være egnet.

Vurdering av elevene

Dette er en introduksjonsaktivitet som ikke nødvendigvis egner seg for vurdering med karakter. Men det er en god anledning til å observere elevens tilnærming til oppgavene og gi tilbakemelding på hvilken vanskegrad eleven bør legge seg på. Eksempelvis er det i mange av oppgavene nyttig å gi tilbakemelding til elever som gjentar kodeblokker i stedet for å bruke løkker.

Ballsprett

Elevene skal lage et program som er en enkel simulering av en ball som slippes fra en viss høyde, og deretter spretter i gulvet. De skal sammenligne simuleringen med fysiske målinger, og justere programmet slik at den kommer så nær virkeligheten som mulig.

Kompetansemål

- Utvikle og feilsøke programmer som løser definerte problemer, inkludert realfaglige problemstillinger, og kontroll eller simulering av fysiske objekter
- Bruke grunnleggende prinsipper i programmering, slik som løkker, tester, variabler, funksjoner og enkel brukerinteraksjon

Læringsmål

- Eleven kan lage et program som simulerer ballsprett og demping
- Elevene kan bruke og forklare løkker, tester og variabler

Utstyr

- Ball og målestokk
- Datamaskin eller nettbrett
- Ulike programmeringsverktøy som Scratch eller Processing

Oppgave til elevene

- Slipp en ball og be elevene beskrive bevegelsen med sprett og demping.
- La dem prøve seg fram med å overføre dette til kode, altså finne ut hvordan de får en datamaskin til å vise en ball som spretter.
- Begynn med en veldig enkel og urealistisk simulering uten tyngdekraft. Selve sprettet kan programmeres ved at farten snus (den nye farten er lik minus den gamle).
- For å få et mer realistisk sprett kan eleven multiplisere farten med en dempingsfaktor (et tall mindre enn 1).
- Tyngdekraft kan simuleres ved hele tiden å trekke fra litt fra hastigheten. Eksempel på ferdig kode er vist lenger nede.
- Ikke gi fasiten for tidlig. Elevene skal få prøve seg fram og ikke være redde for å feile på veien.

Elevene kan så gjøre eksperimenter med en ekte ball og måle hvor høyt ballen faktisk spretter for hvert sprett. De kan da justere akselerasjonen (-0,3 i Scratch-eksempelet) og dempingen (-0.7 i Scratch-eksempelet) så simuleringen ser realistisk ut.

Se eksempel på dette i Scratch her.

Tilpasning til ulike elevgrupper

Dette er et opplegg som vil være gjennomførbart for elever på flere nivåer. For nybegynnere kan det enkelt simuleres i Scratch, men for mer erfarne elever egner Processing seg godt.

Tilpasning til ulike elevgrupper

Programmering av tyngdekraft og sprett kan brukes i mange typer spill. Derfor kan det kan være naturlig å la elevene videreutvikle koden sin til et større spill. Her kan de får frie tøyler, eller de kan følge Kodeklubbens Scratch-prosjekter Lunar Lander og Asteroids eller Processing-prosjekt Ping pong.

Mulige læringsmål og vurdering av høy måloppnåelse i eksempelet ballsprett:

Læringsmål for eleven	Kjennetegn på høy måloppnåelse for eleven
Lage et program som simulerer ballsprett og demping	• lager et program som simulerer sprettet på en realistisk måte, med tyngdekraft og demping i sprettet. Ballen kan sprette mange ganger. • forklarer i hvilken grad simuleringen samsvarer med virkeligheten.
Bruke og forklare løkker, tester og variabler	• bruker løkker på en hensiktsmessig måte for å simulere ballens bevegelse. • bruker tester for å finne ut når ballen treffer gulvet. • bruker variabler for å representere posisjon, fart og akselerasjon. • forklarer hvordan løkker, tester og variabler er brukt i programmet, og på hvilken måte dette bidrar til å gjøre programmet bedre.

Vurdering

Elevene kan vurderes ut fra hvor godt programmet deres simulerer ballsprettet, og hvor godt de muntlig kan forklare bruken av løkker, tester og variabler.

Fra problem til delproblem (arbeidsprosess)

Dette er en åpen oppgave som kan passe for elever med ulik teknisk kompetanse. Her handler det om å ta utgangspunkt i et problem, gjerne reelt, og bryte det ned til delproblemer som kan løses ved hjelp av programmering. Løsningen kan ta form av en nettside, et spill, en app eller en annen programmeringsbasert løsning.

Kompetansemål

- Omgjøre problemer til konkrete delproblemer, vurdere hvilke delproblemer som lar seg løse digitalt, og utforme løsninger for disse
- Utvikle og feilsøke programmer som løser definerte problemer, inkludert realfaglige problemstillinger, og kontroll eller simulering av fysiske objekter

Læringsmål

- gjøre om en definert problemstilling til konkrete delproblemer
- vurdere ulike løsningsmetoder
- utvikle, feilsøke og forbedre programmer
- kontrollere eller simulere fysiske objekter

Utstyr

Elevene kan gjerne benytte tankekartverktøy eller gule lapper i idémyldringen.

Oppgave til elevene

Her skal elevene kunne bryte ned en definert problemstilling i mindre delproblemer for å finne løsninger. Oppgaven egner seg godt til å arbeide i grupper.

- Sørg for at alle elevene har en felles forståelse av hva problemet går ut på.
- La elevene diskutere seg frem til hvilke problemer som lar seg løse ved hjelp av programmering, og designe løsninger for disse problemene.

Arbeidet kan deles inn i fire faser:

- Introduksjon: Forklar hva problemet går ut på, gjerne basert på en reell og kjent problemstilling.
- Nedbryting: Kjør en idémyldring, ved hjelp av for eksempel tankekart, for å finne hvilke mindre delproblemer det store problemet består av.
- Identifisering: Vurder hvilke av delproblemene som kan la seg løse ved hjelp av programmering. Dersom flere delproblemer egner seg, kan de løses av forskjellige grupper.
- Løsning: Foreslå en løsning på problemet. Løsningen kan innebære nettside, spill, app, kontroll eller simulering av et fysisk objekt, eller en annen programmeringsbasert løsning.

Problemstilling (overordnet)	Mulige delproblemer (årsaker)	Mulige løsninger (programmerbar)
Det kommer for få foreldre på foreldremøter.	• Foreldrene glemmer datoen eller tidspunktet. • Foreldrene er ikke interessert i temaet. • Tidspunktet passer ikke for mange.	• Lage en app til foreldrene som varsler dem om kommende møter. • Lage nettside med avstemninger om tema og tidspunkt.
Elevene skårer dårlig på matematikkeksamen.	Mange elever sliter med å forstå brøkregning.	Lage et spill som gir elevene forståelse av brøkregning.
Skolens resepsjon er stadig opptatt og har ikke tid til å hjelpe elevene.	Foreldrene ringer for å spørre om informasjon rundt skolens arrangementer.	Lage en nettside som oppdateres med informasjon om kommende arrangementer.
Skolen har et akvarium, som alle klassene bytter på å ha ansvaret for. Men fiskene dør altfor ofte.	• Fiskene får ikke mat regelmessig nok, fordi klassene glemmer det eller noe kommer i veien. • Fiskene får mat for ofte. • Vannet har feil temperatur.	• Programmere en robotarm som gir fiskene mat til faste tider. • Programmere en sensor som måler vanntemperatur og sender SMS når noe må gjøres.

Tilpasning til ulike elevgrupper

Elevene kan selv komme opp med problemer de vil løse. Problemene og delproblemene kan ha ulik kompleksitet, og det samme gjelder løsningsforslagene.

Prosessen kan være lærerstyrt i ulik grad. For eksempel kan det settes rammer og begrensinger for elevene ved at læreren spesifiserer elementer som må være med i en løsning.

En åpen oppgave med begrensninger kan være å be elevene om å komme opp med et problem i hverdagslivet som skal løses ved bruk av en bestemt teknologi. Eksempel på en slik oppgave: Elevene skal løse et vanlig problem som har med trafikk å gjøre, og designe en løsning for problemet som benytter mikrokontrolleren Arduino.

Vurdering

- Elevene kan dokumentere problemløsningsprosessen på ulike måter: tankekart, bilder av en kreativ prosess, m.m.
- De kan lage en skisse til løsning (på papir eller i et tegne- eller designprogram). Gruppene kan presentere muntlig for hverandre hvordan de har jobbet med prosessen, hvilket delproblem de bestemte seg for å løse, ulike løsninger de vurderte, og hvilken løsning de til slutt valgte.

En slik presentasjon kan gjerne gjøres underveis slik at elevene kan få innspill fra hverandre i prosessen med å bryte ned problemet eller designe løsninger.

Elevene kan vurderes på hvor godt de argumenterer for sine valg.

Marslanding

Oppdraget er å simulere en landing av roveren Curiosity på Mars. Curiosity skal lande trygt og kjøre videre etter landing. Denne oppgaven kan gjerne gjøres som et større prosjekt som elevene kan arbeide med over tid. Læreren kan tilpasse omfanget av prosjektet, avhengig av tilgjengelig tid, ambisjonsnivå og elevenes kompetanse.

Kompetansemål

- Omgjøre problemer til konkrete delproblemer, vurdere hvilke delproblemer som lar seg løse digitalt, og utforme løsninger for dem
- Overføre løsninger til nye problemer ved å generalisere og tilpasse eksisterende programkode og algoritmer
- Bruke grunnleggende prinsipper i programmering, slik som løkker, tester, variabler, funksjoner og enkel brukerinteraksjon
- Dokumentere og forklare programkode gjennom å skrive hensiktsmessige kommentarer og ved å presentere egen og andres kode
- Utvikle og feilsøke programmer som løser definerte problemer, inkludert realfaglige problemstillinger og kontroll eller simulering av fysiske objekter

Læringsmål

- Elevene kan bruke løkker, tester og variabler
- Elevene kan finne kreative løsninger på et gitt problem
- Elevene kan lage et program som simulerer en marslanding i programmeringsspråket som er valgt
- Elevene kan gjenbruke og tilpasse tidligere kode

Utstyr

- Datamaskin eller nettbrett
- Dersom skolen har tilgang til det, kan ulike roboter eller droner benyttes

Oppgave til elevene

Denne oppgaven kan bli ganske kompleks, så vi anbefaler at elevene jobber gruppevis. For at alle skal få en felles forståelse av hva oppgaven går ut på, bør det være en felles oppstart. Elevene ser video av landing på Mars - det finnes mange videoer tilgjengelig på nettet. Deretter går lærer og elever sammen gjennom oppgaven som skal løses, og diskuterer hvordan de kan gå frem for å løse den.

Gruppearbeid:
Undersøke gravitasjon på Mars. Hvilke egenskaper har Curiosity? Hvilket oppdrag skal den utføre? Hvordan har de testet roboten før oppskytning? Lage en løsning gruppevis.

Krav til løsningen:
Curiosity skal lande trygt på Mars og kjøre videre etter landing. Det forutsetter noen beregninger rundt elementer som slipphøyde, gravitasjon og hastighet ved landing.

Tilpasning til ulike elevgrupper

Det kan være lurt å skille mellom hvilket nivå elevene er på, og eventuelt jobbe med temaet i flere omganger med økende vanskegrad. For eksempel kan elevene som har faget første gang, og elever som fortsatt trenger å øve på blokkbasert programmering, settes sammen i grupper som skal løse oppgaven i et blokkbasert programmeringsspråk (f.eks. Scratch). Viderekomne elever kan settes sammen i grupper som skal løse oppgaven i et tekstbasert programmeringsspråk (f.eks. Processing eller Python).

Vanskegraden kan justeres gjennom å stille ulike krav til utregningene som gjøres. For de mest uerfarne kan det være nok å stille krav til maks hastighet ved å bruke en enkel gravitasjonssimulering, og f.eks. legge til fallskjermer eller bremseraketter for å justere farten. Senere kan man øke vanskegraden ved å legge til faktorer som luftmotstand og varmeutvikling. Viderekomne elever kan gjenbruke og tilpasse koden fra et annet programmeringsspråk enn det de presenterer løsningen sin i.

En mulig videreføring av oppgaven kan være å finne alternative måter elevene kan utføre landingen på. Underveis kan det være nyttig at elevene presenterer programmene sine, og begrunner valg de har tatt underveis. Dette åpner for at andre kan komme med innspill, og at elevene får idéer til mulige forbedringer i programmet sitt.

Vurdering

Vurderingen kan med fordel omfatte både prosessen og det ferdige resultatet. Gjennom å la elevene presentere programmene sine underveis, kan de også få innspill som kan hjelpe dem til å forbedre koden sin. Det vil uansett være hensiktsmessig å la elevene presentere programmet sitt til slutt for å kunne begrunne de

valgene som er tatt.

Robotstøvsuger

Oppdraget er å utforske og utvikle en modell for en robotstøvsuger, og å designe programmer for å simulere eller kontrollere robotstøvsugerens bevegelser i et rom. Oppdraget kan gjøres avgrenset eller avansert, avhengig av tilgjengelig tid og elevenes nivå. Beskrivelsen vektlegger hvordan elevene kan arbeide med hovedområdet modellering. Til modellering hører kunnskap om hva slags problemer som egner seg for å løses av en datamaskin, hvordan disse kan brytes ned i delproblemer, og hvordan løsninger kan utformes.

Kompetansemål

- Omgjøre problemer til konkrete delproblemer, vurdere hvilke delproblemer som lar seg løse digitalt, og utforme løsninger for dem
- Bruke flere programmeringsspråk der minst ett er tekstbasert
- Bruke grunnleggende prinsipper i programmering, slik som løkker, tester, variabler, funksjoner og enkel brukerinteraksjon
- Utvikle og feilsøke programmer som løser definerte problemer, inkludert realfaglige problemstillinger og kontroll eller simulering av fysiske objekter
- Overføre løsninger til nye problemer ved å generalisere og tilpasse eksisterende programkode og algoritmer

Læringsmål:

- Kan definere oppdraget til en robotstøvsuger
- Eleven kan forklare hvilke delproblemer som må løses av robotstøvsugeren, slik som navigasjon, søking, støvsuging, unngå hindringer, returnere til ladestasjon, osv.
- Kan finne mulig tekniske løsninger på et praktisk problem
- Kan utvikle programkode og algoritmer som løser problemet

Utstyr

- Datamaskin eller nettbrett
- Ulike programmeringsverktøy og språk kan benyttes for å simulere støvsugeren, f.eks. Scratch, Python eller JavaScript
- Dersom skolen har tilgang til det, kan ulike roboter eller droner med fordel benyttes (Lego NXT, Spheros, Arduino eller Raspberry Pi) og med tilhørende programvare og apper for å programmere dem

Oppgave til elevene

Innledende oppgave i smågrupper:

Elevene utforsker og definerer egenskapene en robotstøvsuger bør ha. Her er det mulig å ta utgangspunkt i hensikten med en robotstøvsuger, og se på de ulike modellene som eksisterer på markedet. Spørsmål som kan hjelpe elevene i gang med arbeidet:

- Hvilke egenskaper må en robotstøvuger ha, og hvilke kan den ha?
- Finnes det forskjellige løsninger på hvordan robotstøvsugere kan utføre oppdraget sitt?
- Hvordan sikre at støvsugeren dekker hele det aktuelle arealet? (Må den støvsuge hele arealet hver gang?)
- Hva kan den enkleste løsningen være?

I arbeidet med å definere egenskapene til robotstøvsugeren kan læreren be elevene om å skille mellom problemer som kan løses ved hjelp av programmering, slik som kontroll og styring av roboten, og problemer som må løses av design og utforming av den fysiske roboten (for eksempel motor og hjul som lar roboten bevege seg rundt i rommet). De kan selv definere hvilke overordnede oppgaver roboten må løse, og så dele disse inn i mindre problemer som de kan programmere løsninger for.

Lage skisseprogram:

Elevene kan lage et program (for eksempel i Scratch) som illustrerer bevegelsene til en robotstøvsuger sett ovenfra. Underveis kan elevene presentere ulike løsninger for klassen, og diskutere fordeler og ulemper med disse. Hvilke algoritmer er mest hensiktsmessige for å sikre støvsuging av hele rommet? Hvilke vil gjøre at robotstøvsugeren bruker kortest mulig tid på arealet? Hva kan være beste løsningen for en mest mulig effektiv støvsuging? Hvordan kan koden skrives enklest mulig for dette?

En mulig organisering er at grupper av elever jobber med å løse ulike problemer, f.eks. kan noen arbeide med styring av roboten, noen med hvordan den skal bevege seg mest effektivt i et rom, og noen med hvordan den skal unngå å falle ned trapper, osv.

Mulig utvidelse: programmere en fysisk robot

Bruk erfaringene fra utviklingen av skisseprogrammet til å programmere en fysisk robot (for eksempel LEGO-robot eller Sphero) til å jobbe som en robotstøvsuger. Elevene kan teste programkoden sin i praksis og utføre forbedringer basert på observasjoner og tidtakinger.

Elevene kan lage en prototype av robotstøvsugeren med eget design. De kan ta hensyn til brukervennlighet, og se på praktiske løsninger knyttet til for eksempel opplading av roboten og til samling og tømming av støv. Hvordan skal roboten oppdage hindringer, og hvordan komme utenom disse. Kanskje dette er utfordringer det kan finnes programmerbare løsninger på?

Tilpasning til ulike elevgrupper

Oppdraget åpner for en videreføring der elevene kan implementere løsningen sin. Dette kan gjøres på ulike måter: som en simulering i Scratch, ved hjelp av tekstbaserte språk, eller ved å programmere fysiske roboter og til og med bygge robotene selv.

Problemene som elevene skal finne løsninger på, har ulik vanskegrad. Det kan for eksempel være enklere å skrive koden som gjør at roboten snur når den møter en hindring, enn å finne den optimale algoritmen for å rengjøre et rom effektivt. Videre kan kompleksiteten økes ved å tildele arealer med ulik vanskegrad på det støvsugeren skal rengjøre. Et umøblert, kvadratisk rom er enklere enn et møblert rom med krinker og kroker.

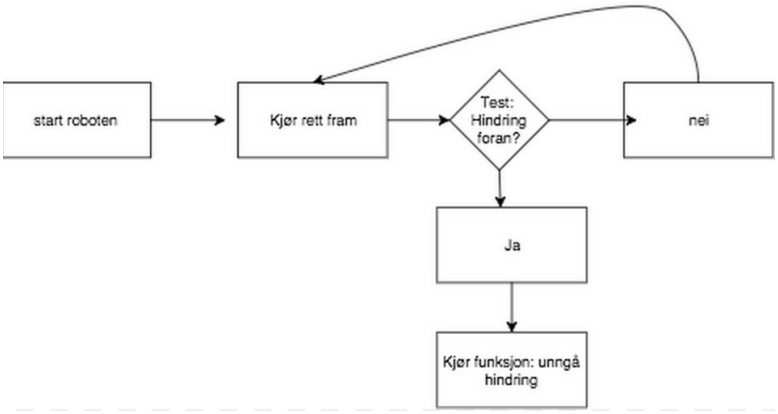
Det går også an å bygge en programmerbar robot 'fra bunnen av' med Raspberry PI eller arduino, og utforme skroget i tre eller 3D-print. Et prosjekt av denne størrelsen bør trolig kombineres med aktiviteter i andre fag, for eksempel med naturfag, matematikk og kunst- og håndverk.

Vurdering

Denne oppgaven kan bli meget omfattende med mange involverte deloppgaver. Det kan derfor være aktuelt å gjennomføre vurderingen i flere etapper, der de enkelte stegene i prosessen vurderes for seg. Blant elementene som kan vurderes, er hvor godt elevene har klart å bryte ned problemet i delproblemer, hvor godt de begrunner valgene av løsning, hvor effektivt løsningen deres utfører oppdraget, og hvorvidt de klarer å benytte seg av egen og andres eksisterende programkode i implementeringen.

Elevene kan lage skisser av hvordan robotstøvsugere kan forflytte seg over arealet den skal støvsuge. De kan dokumentere modelleringsprosessen ved bilder, presentasjoner, tekst og muntlige presentasjoner.

Elevene bør lage en form for kravspesifikasjon for robotstøvsugeren sin som beskriver hvilke oppgaver den skal utføre. Kravspesifikasjonen kan med fordel inneholde beskrivelser av ønsket funksjonalitet, vist gjennom flytdiagram eller pseudo-kode (kode beskrevet i naturlig språk, f.eks. "Hvis hindring foran; unngå hindring. Ellers: Kjør rett frem").



Eksempel på flytdiagram for hvordan en robot skal fungere.

Vurderingen kan gis på grunnlag av kvaliteten på den foreslåtte løsningen, og på hvor godt elevene kan begrunne sine valg. Det kan være gode grunner til å velge bort komplekse løsninger for en mer gjennomførbar løsning.

Kompetansemål (eleven skal kunne)	Læringsmål (eleven skal kunne)	Kjennetegn på høy måloppnåelse (eleven)
Omgjøre problemer til konkrete delproblemer, vurdere hvilke delproblemer som lar seg løse digitalt, og utforme løsninger for disse	• Definere oppdraget til en robotstøvsuger. • Forklare hvilke del-problemer som må løses av robotstøvsugeren, slik som navigasjon, søking, støvsuging, unngå hindringer, returnere til ladestasjon, osv.	• Forklarer hvilke ulike oppgaver en god robotstøvsuger må løse. • Forklarer hvilke av disse oppgavene som kan løses ved hjelp av programmering, og hvilke oppgaver som må løses på annen måte (for eksempel design og utforming av selve roboten).
Overføre løsninger til nye problemer ved å generalisere og tilpasse eksisterende programkode og algoritmer.	Utvikle programkode og algoritmer som løser problemet	• Har laget en programkode med ryddige kommentarer. • Presenterer og forklarer programkoden som skal kunne løse problemet. • Peker på hvilke deler av programmet som baserer seg på tidligere programkode, og begrunner valg av gjenbruket. • Begrunner valg av løsninger, og ser løsningen i lys av andre mulige løsninger. • Diskuterer mulighetene for at koden kan effektiviseres og videreutvikles.
Bruke grunnleggende prinsipper i programmering, slik som løkker, tester, variabler, funksjoner og enkel brukerinteraksjon.	Utvikle programkode og algoritmer som løser problemet	• Har laget en programkode med ryddige kommentarer. • Presenterer og forklarer programkoden som skal kunne løse problemet.

		• Peker på hvilke deler av programmet som baserer seg på tidligere programkode, og begrunner valg av gjenbruket. • Begrunner valg av løsninger, og ser løsningen i lys av andre mulige løsninger. • Diskuterer mulighetene for at koden kan effektiviseres og videreutvikles.
Dokumentere og forklare programkode gjennom å skrive hensiktsmessige kommentarer og ved å presentere egen og andres kode.	Utvikle programkode og algoritmer som løser problemet	• Har laget en programkode med ryddige kommentarer. • Presenterer og forklarer programkoden som skal kunne løse problemet. • Peker på hvilke deler av programmet som baserer seg på tidligere programkode, og begrunner valg av gjenbruket. • Begrunner valg av løsninger, og ser løsningen i lys av andre mulige løsninger. • Diskuterer mulighetene for at koden kan effektiviseres og videreutvikles.
Utvikle og feilsøke programmer som løser definerte problemer, inkludert realfaglige problemstillinger og kontroll eller simulering av fysiske objekter.	Finne mulig tekniske løsninger på et praktisk problem	Eleven skisserer løsninger for å programmere roboten innenfor de begrensinger som settes av utstyr og programvare

Spillprosjekt

I denne oppgaven skal elevene få lage sitt eget spill. Her blir det spesielt lagt vekt på å kommentere egen kode og å presentere den for andre.

Kompetansemål

Dokumentere og forklare programkode gjennom å skrive hensiktsmessige kommentarer og ved å presentere egen og andres kode.

Læringsmål

- Kan skrive hensiktsmessige kommentarer til egen kode
- Kan lese og forstå medelevers kode
- Kan formidle nyttige tilbakemeldinger på medelevers programmer
- Kan presentere egen kode og forklare hvordan programmet virker
- Kan forbedre egenutviklet program på grunnlag av tilbakemeldinger fra medelevene og læreren

Utstyr

- Datamaskin eller nettbrett
- Ulike programmeringsverktøy, for eksempel Scratch, Kodu og MIT App Inventor

Oppgave til elevene

- Bli enige om hva som kjennetegner et bra spill, gjerne gjennom en idémyldring som munner ut i en konkret skisse av hvordan spillet skal være.
- Bli enige om hvilke deloppgaver de ulike gruppemedlemmene kan jobbe med. På dette stadiet kan det være naturlig å veilede noen av gruppene til å velge et passe ambisiøst spill, eventuelt å tilpasse et spill de har laget etter oppskrift før.
- Lag en forenklet utgave av spillet de vil lage, for så å bygge det ut når de ser at det fungerer. Pass på at elevene kommenterer koden sin slik at det blir forståelig for dem selv og andre i ettertid.
- La elevene teste og gi tilbakemeldinger på andres spill, og også se koden og lese kommentarene. Læreren kan også komme med innspill og tilbakemeldinger til elevenes spill.
- La elevene forbedrer spillet med utgangspunkt i tilbakemeldingene.
- Til slutt presenterer elevene det ferdige spillet.

Tilpasning til ulike elevgrupper

Et slik spillprosjekt kan gjennomføres med mange ulike verktøy til mange ulike plattformer. Som en start kan Scratch være enkelt, mens for 3D-grafikk kan Kodu være aktuelt. Mer viderekomne brukere kan lage en app i MIT App Inventor.

Vurdering

Det er naturlig å vurdere presentasjonen av egen kode i den avsluttende presentasjonen, med spesiell vekt på forståelighet og god bruk av kommentarer i koden.

Elever som ressurs

En sentral del av programmererkulturen er å dele sin kompetanse og å være mentorer for andre. Faget kjennetegnes ved at det er et nærmest uendelig antall muligheter for variasjon og spesialisering (ulike språk, ulike plattformer, ulik teknologi, osv.). En lærer verken kan eller skal være ekspert på alt, men elevene bør oppfordres til å dele sin kompetanse. Elevene kan være en ressurs for hverandre ved å holde presentasjoner, gi veiledning til medelever og utvikle veiledningsmateriell til andre i klassen.

Kompetansemål

- Dokumentere og forklare programkode gjennom å skrive hensiktsmessige kommentarer og ved å presentere egen og andres kode
- Bruke grunnleggende prinsipper i programmering, slik som løkker, tester, variabler, funksjoner og enkel brukerinteraksjon
- Utvikle og feilsøke programmer som løser definerte problemer, inkludert realfaglige problemstillinger og kontroll eller simulering av fysiske objekter

Læringsmål

- Elevene kan forklare og begrunne egen kode
- Elevene kan bruke og forklare løkker, tester, variabler, funksjoner og enkel brukerinteraksjon
- Elevene kan feilsøke egen og andres kode

Utstyr

- Datamaskin eller nettbrett

Oppgave til elevene

Et godt samarbeid og en sterk delingskultur i klassen kan bidra til at flere elever lærer mer, samt at elevene

får utfordringer tilpasset sitt nivå. Her er noen eksempler på hvordan elevene kan være en ressurs for hverandre:

- Elevene kan holde korte presentasjoner av eget arbeid for klassen eller mindre grupper. Da får de trening i å vurdere egen kode og forklare den. Dette åpner også for at elevene sammen kan diskutere mulige forbedringer i koden, i tillegg til at elevene kan inspirere hverandre med ulike løsninger.
- Elevene kan være veiledere for hverandre. For eksempel kan elevene settes i par, hvor de skal bistå hverandre med utvikling og feilsøking av programmene de har laget. Det er også mulig at elever kan ta rollen som veiledere dersom det jobbes med programmeringsspråk som de har erfaring med.
- Elevgrupper kan få i oppdrag å presentere spesifikke temaer eller konsepter innen programmering, for eksempel løkker. Elevene kan presentere hvordan løkker ser ut i både blokkbaserte og tekstbaserte programmeringsspråk, og så sammenlikne dem. De kan også utfordres til å komme opp med metaforer og sammenlikninger for å forklare et konsept innen programmering, for eksempel kan en algoritme sammenliknes med en matoppskrift. Presentasjonene kan vises i klassen, eller de kan spilles inn som instruksjonsvideoer til nytte for andre elever.
- Deling og gjenbruk av programkode er svært vanlig innen programmeringsfaget. Elevene kan introduseres for dette fellesskapet gjennom at de oppfordres til å dele sine løsninger og kommentere på andres. Da kan de også få tilbakemeldinger fra personer utenfor skolen, noe som kan virke ekstra motiverende for enkelte elever. Deling av egenutviklede programmer kan gjøres enkelt via f.eks. Scratch eller via steder som GitHub som brukes av amatører og profesjonelle programmerere over hele verden.
- Elevene kan undervise elever som ikke tar programmeringsfaget, for eksempel yngre elever. Stadig flere skoler gjennomfører en årlig 'kodetime' for alle elever, der målet er å programmere i en skoletime (se eksempel om Kodetimen). I Oslo kommune skal ungdomstrinnselever undervise elever på 4. trinn i programmering på Aktivitetskolen/SFO. Elever som tar programmeringsfaget, kan få i oppdrag å gjennomføre en kodetime med elever på egen skole eller på en skole i nærheten. Dette kan gjerne gjøres i samarbeid med den nærmeste barneskolen.

Tilpasning til ulike elevgrupper

Elever med lite erfaring kan utfordres til å dele sine prosjekter og programmer med medelever og omverdenen, for eksempel gjennom å publisere på delingsarenaer i Scratch eller GitHub. De kan også bli bedt om å formulere presist hva de ønsker å oppnå, og hva de eventuelt trenger veiledning om. Mer erfarne elever kan utfordres til å bidra til programmeringsfellesskapet, enten ved å kommentere og bidra til videreutvikling av medelevers prosjekter, eller ved å delta i et større fellesskap på internett.

Erfarne elever kan være mentorer for medelever innen områder de har erfaring med. Det er viktig å ha en dialog med elevene og sørge for at de er motiverte til oppgaven og trygge på opplegget som skal gjennomføres. Elevene skal ikke være "ubetalte ekstralærere", og en slik mentorordning må derfor designes

slik at den bidrar positivt til opplæringen for de elevene som er mentorer, og at det er tydelig hvilke læringsmål som gjelder for dem.

Vurdering

Elevene kan vurderes på hvordan de forklarer kode eller konsepter i programmering i presentasjoner for andre elever. De kan også vurderes på hvordan de feilsøker andres kode, og på hvordan de gir tilbakemeldinger når de veileder andre. De kan også vurderes på hvordan de gir tilbakemelding til andre programmerere, både innad i klassen og andre steder. Elevene kan lenke til sine delte prosjekter, ta skjermbilder eller reflektere muntlig eller skriftlig om hvordan det var å dele eget eller kommentere andres arbeid.

Vurdering

Læringsmål for ulike oppgaver og oppdrag kan utformes av læreren i forkant, av elevene selv eller som et samarbeid mellom læreren og elevene.

Her er eksempler på hvordan læringsmål kan utformes på bakgrunn av kompetansemål og beskrivelse av kjennetegn på høy måloppnåelse. Eksemplene er tenkt brukt til underveisvurdering.

Kompetansemål (Eleven skal kunne)	Læringsmål (Eleven kan)	Kjennetegn på høy måloppnåelse (Eleven)
Gjøre rede for hvordan datamaskiner og programmer fungerer, inkludert et utvalg utbredte programmeringsspråk og deres bruksområder	Gjøre rede for hvordan en datamaskin fungerer	Forklarer i grove trekk hvordan en datamaskin fungerer, og hva som er forskjellen på inn-enhet og ut-enheter.
	Gjøre rede for hvordan et program fungerer	• Forklarer forskjellen og sammenhengen mellom programvare og maskinvare. • Forklarer hva programmeringsspråk er, hva maskinspråk er, hva en kompilator er, og sammenhengen mellom disse.
	Gjøre rede for bruksområdene til ulike programmeringsspråk	• Presenterer ulike programmeringsspråk og deres viktigste bruksområder. • Forklarer hva et tekstbasert programmeringsspråk er.

		• Viser eksempler på hvordan ulike programmeringsspråk kan brukes til å utføre samme type oppgave.
Omgjøre problemer til konkrete delproblemer, vurdere hvilke delproblemer som lar seg løse digitalt, og utforme løsninger for disse	Omgjøre et problem til konkrete delproblemer	• Deler opp problemet på en hensiktsmessig måte. • Formuler konkrete delproblemer.
	Vurdere hvilke delproblemer som lar seg løse digitalt	• Vurderer de ulike delproblemene for å finne mulige digitale løsninger på dem • Peker på muligheter og begrensinger med digital løsning for utvalgte delproblem
	Utforme løsninger for ulike delproblemer	• Utformer hensiktsmessige løsninger på delproblemer. • Forklarer hvordan valgt løsning vil fungere og hvilke behov den løser. • Reflekterer over realismen i valgt digital løsning, inkludert hvor vanskelig det vil være å utvikle. • Argumenterer godt for sin valgte løsning og vurderer denne opp mot andre mulige løsninger.
Dokumentere og forklare programkode gjennom å skrive hensiktsmessige kommentarer og ved å presentere egen og andres kode	Skrive hensiktsmessige kommentarer til programkode	• Skriver beskrivende kommentarer i egen kode slik at den også kan forstås av andre. • Viser eksempler på gode (hensiktsmessige) og mindre gode kommentarer i egen eller andres kode. • Tar utgangspunkt i egen eller andres kode og forbedrer denne

		gjennom å skrive gode kommentarer som forklarer hva de ulike delene av koden gjør.
	Presentere og forklare egen kode	• Gjør rede for hvordan egen kode er bygd opp. • Forklarer hva koden gjør.
	Forstå og presentere kode andre har skrevet	Bruker andres kommentarer til å forstå og forklare hva koden gjør

Hovedområdet koding - eksempler læringsmål og kjennetegn

Kompetansemål (Eleven skal kunne)	Læringsmål (Eleven kan)	Kjennetegn på høy måloppnåelse (Eleven)
Bruke flere programmeringsspråk der minst ett er tekstbasert	Bruke flere programmeringsspråk der minst ett er tekstbasert	• Bruker flere programmeringsspråk, der minst ett er tekstbasert, til å lage programmer. • Lager programmer som løser oppgaven/problemet på en god måte. • Lager programmer som virker etter intensjonen • Velger mellom flere programmeringsspråk for å løse en oppgave og forklarer hvorfor dette språket ble valgt framfor et annet.
Bruke grunnleggende prinsipper i programmering, slik som løkker, tester, variabler, funksjoner og enkel brukerinteraksjon	Bruke løkker	• Forklarer hva en løkke er, og gir konkrete eksempler på dette. • Viser til eksempler som nyttegjør seg av ulike typer

		løkker (f.eks. for-løkker og while-løkker), og forklarer forskjellen på disse. • Benytter løkker for å effektivisere programkoden, når programmet skal gjøre gjentakelser.
	Bruke tester	• Forklarer hva en test er, og gir konkrete eksempler på ulike tester. • Viser til eksempler hvor det benyttes tester (f.eks. if-setninger og if-else-setninger) • Benytter tester der det er hensiktsmessig i programmet.
	Bruke variabler	• Forklarer hva en variabel er, og at variabler kan representere ulike datatyper, for eksempel tekst og tall. • Viser konkrete eksempler på variabler, og forklarer hvorfor de brukes i programmer. • Benytter seg av variabler i programmet, og har navngitt dem på en hensiktsmessig måte. • Demonstrerer hvordan variabler fungerer i egenutviklet program
	Bruke funksjoner	• Forklarer hva en funksjon er, og

		• gir konkrete eksempler på dette. • Forklarer hvorfor vi bruker funksjoner i programmer. • Bruker funksjoner i programmet på en hensiktsmessig måte.
	Lage enkel brukerinteraksjon	• Forklarer hva brukerinteraksjon er, og gir konkrete eksempler på dette. • Implementerer brukerinteraksjon og forklarer hvordan brukere kan navigere i programmet. • Argumenterer for valg av løsninger for brukerinteraksjonen, og tar hensyn til brukervennlighet.
Utvikle og feilsøke programmer som løser definerte problemer, inkludert realfaglige problemstillinger og kontroll eller simulering av fysiske objekter	Utvikle og feilsøke programmer som løser definerte problemer	• Utvikler et program som løser et definert problem på en god måte. • Kan oppdage feil i egen eller andres kode, kan lete etter og finne den, samt korrigere feilen. • Beskriver utviklingsprosessen av programmet, inkludert idéfase, koding, testing, feilsøking og forbedring.
	Utvikle og feilsøke programmer som løser realfaglige problemstillinger	• Utvikler et program som løser et definert realfaglig problem på en god måte. • Gjør rede for hvordan

		løsningen samsvarer med virkeligheten, forklarer hvilke forenklinger som er gjort, og argumenterer for dette. • Kan oppdage feil i egen eller andres kode, kan lete etter og finne den, samt korrigere feilen. • Vurderer eget program i forhold til om det er en god løsning på den realfaglige problemstillingen, og foreslår mulige forbedringer • Beskriver utviklingsprosessen av programmet, inkludert idéfase, koding, testing, feilsøking og forbedring.
	Utvikle og feilsøke programmer som kontrollerer eller simulerer fysiske objekter	• Utvikler et program som kontrollerer eller simulerer fysiske objekter på en god måte. • Gjør rede for hvordan løsningen samsvarer med virkeligheten. • Kan oppdage feil i egen eller andres kode, kan lete etter og finne den, samt korrigere feilen. • Beskriver utviklingsprosessen av programmet, inkludert idéfase, koding, testing, feilsøking og forbedring.
Overføre løsninger til nye problemer	• Overføre løsninger til	• Drøfter hvordan

ved å generalisere og tilpasse eksisterende programkode og algoritmer	nye problemer. • Utvikle programmer som bygger på og gjenbruker deler av andre programmer til å løse liknende problemer	eksisterende programmer kan benyttes til å løse nye problemer. • Tilpasser eksisterende programkode til å løse nye problemer. • Forklarer hvordan egenutviklet program bygger på del-løsninger fra tidligere programmer • Drøfter hvordan deler av egenutviklet program kan gjenbrukes i andre programmer
	Gjenbruke og tilpasse egen og andres kode	• Lager et nytt program som bygger på egen eller andres programkode. • Viser at det nye programmet bygger på eksisterende programkode. • Gjør rede for tilpasninger og generaliseringer som er utført.

Støttemateriell

Delingsarenaen iktipraksis.no

Iktipraksis.no er en arena for lærere til å dele sine pedagogiske opplegg knyttet til digitale verktøy. På siden er det egne opplegg knyttet til koding og programmering, hvor det er konkrete tips og veiledninger til hvordan en eller flere undervisningstimer kan planlegges og gjennomføres.

MOOC: valgfag programmering

Senter for IKT i utdanningen laget en mooc (nettbasert kurs) som en åpen ressurs for gi lærere en innføring i

programmering og inspirerende ressurser til hvordan du kan innføre og bruke programmering i din undervisning. Her vil lærere få innføring i noen grunnleggende begreper og prinsipper i programmering og konkrete tips til hvordan de kan gjennomføre undervisning i faget - støttet med gode eksempler på bruk av teknologi.

Kurset er bygd opp med fagmoduler som handler om faget, og noen moduler som kan brukes som oppslagsverk. I tillegg vil det bli utviklet innføringskurs i noen utvalgte programmeringsspråk. MOOCen finner du på vår kursportal.

Kodeklubbens oppgaveside

Kodeklubben har gjennom flere år med undervisning i programmering laget mange veiledninger til flere programmeringsspråk, som fritt kan brukes til undervisning i klasserommet. Veiledningene er på norsk, og beskriver stegvis hvordan man lager et gitt program, for eksempel et spill, en app eller en nettside. Oppleggene er kategorisert i ulik vanskegrad, fra introduksjon til ekspert. Det er også tilknyttet oppgaver på engelsk, samt videoressurser som kan være nyttige.

Lær Kidsa Koding sine skolesider

Lærere som er en del av nettverket Lær Kidsa Koding, har delt mange av sine erfaringer og opplegg på kidsakoder.no/skole. Her finner didaktiske opplegg, årsplaner med programmering, tid/sted for kurs/konferanser og en facebook-gruppe for faglige diskusjoner og erfaringsdeling. Det er også opprettet en egen side med fokus på valgfaget.

Anbefalte språk og utstyr

Programmering valgfag kan gjennomføres på mange ulike måter og med mange typer utstyr. Valgfaget kan gjennomføres uten ekstra investeringer i programvare og utstyr for skolen. Det skal i utgangspunktet være nok med datamaskiner med Internett-tilgang.

Nedenfor beskriver vi et utvalg programmeringsspråk det kan være greit å begynne med, og noe utstyr som kan være nyttig dersom dere har, eller vurderer å anskaffe dette. Hvilke språk og utstyr dere velger å benytte på deres skole, bør selvsagt bygge på hva slags kompetanse og utstyr dere har tilgjengelig hos dere.

Språk og utstyr til blokkbasert programmering

Scratch

Scratch er et grafisk programmeringsspråk som er laget for at barn og unge enkelt skal lære seg programmering. Scratch består av klosser som settes sammen til større blokker, hvor hver kloss representerer en instruksjon. Klossene settes sammen ved å dra og slippe de på skjermen, og programmering i Scratch er litt som å bygge med LEGO. Scratch er oversatt til norsk og er helt gratis å bruke, både offline og online.

App Inventor

App Inventor er et programmeringsmiljø som brukes for å lage apper til Android-telefoner og nettbrett. Som Scratch er App Inventor også klosser med instruksjoner som eleven drar og setter sammen, både når eleven programmerer og designer appen. App Inventor brukes i nettleseren, og siden er på engelsk. En overfører appen til telefonen eller nettbrettet ved å scanne en QR-kode fra skjermen.

Kodu

Kodu er både et dataspill og et læringsverktøy for spillprogrammering i 3D. Ved bruk av bildeikoner programmerer spilleren karakterer, objekter, regler, bestemte handlingsmønstre, hvordan vinne og tape, poengsystemer og digitale naturlover. Kodu er en plattform som er på engelsk, og den fungerer kun på Microsoft-enheter.

Språk og utstyr til tekstbasert programmering

Python

Python er et tekstbasert programmeringsspråk som ble utviklet tidlig på 90-tallet, og er ofte anbefalt til nybegynnere på grunn av språkets enkle syntaks sammenlignet med en del andre tekstbaserte programmeringsspråk. Python passer perfekt for de som ønsker å lære seg programmering, og kan blant annet brukes til 2D-spill, automatisering, vitenskapelige analyser, GUI-applikasjoner og servere.

Processing

Processing
er et tekstbasert programmeringsspråk som er basert på Java. Det ble laget for at det skulle være enkelt for designere og andre som ikke har programmert før, å lage grafiske programmer. Dette gjør Processing perfekt for de som har lyst til å lære å lage seg spill, animasjoner, simuleringer og multimediaprogrammer.

Web (HTML, CSS, JavaScript)

Web består av mange byggesteiner og dersom en skal lage egne nettsider, så må en lære om tre av disse byggesteinene: HTML, CSS og JavaScript. HTML er tekst som beskriver innholdet til en nettside. CSS er stilen, altså hvordan nettsiden skal se ut. Og JavaScript er et programmeringsspråk som forteller hva nettleseren din skal gjøre. Kombineres disse tre byggesteinene riktig, ja da får du en nettside!

Ulike typer maskinvare som kan benyttes

LEGO Mindstorms

LEGO Mindstorms er et produkt fra leketøysprodusenten Lego som gjør det mulig å lage og programmere sin egen robot. Programmeringen skjer ved å sette sammen blokker med instruksjoner. Roboten kan kobles til flere forskjellige sensorer, og i tillegg løper det ut porter hvor det kan kobles til motorer, lamper, osv.. Programmene overføres vanligvis til roboten via en USB-kabel.

Arduino

Arduino er en plattform som består av en rekke forskjellige mikrokontrollere, samt et programmeringsverktøy som heter Arduino IDE. Ved bruk av en Arduino kan du koble sammen ledninger og gi strøm til lys, mikrofoner, høgtalere, sensorer, motorer og andre ting. En Arduino programmeres vanligvis i C, men det er også mulig å styre den i Scratch.

Rapsberry Pi

Raspberry Pi er en liten mini-datamaskin på størrelse med et kredittkort. Maskinen har HDMI, USB og Ethernet innganger og kjører en version av Linux som heter Raspian. I tillegg har Raspberry PI et sett med eletroniske kontakter som kan brukes til å lese og styre eletriske kretser. På en Raspberry Pi kan eleven altså programmere i andre programmeringsspråk, og knytte dette opp mot den fysiske verden.

Ordliste

Algoritmer

Med begrepet algoritme mener vi en oppskrift som forteller oss trinnvis hva som skal gjøres for at noe skal bli fullført - hva som må gjøres for å nå målet. Utfordringen med algoritmer er at de må skrives og følges nøyaktig - og i riktig rekkefølge - for at vi skal få det resultatet vi forventer. Å skrive i riktig rekkefølge er derfor helt essensielt, i tillegg til at man må ta med alle trinnene. En datamaskin følger de instruksjoner den får I den rekkefølge de gis. Et dataprogram er en algoritme, og et dataprogram består ofte av mange algoritmer. Men ikke alle algoritmer er dataprogram.

Brukerinteraksjon

Brukerinteraksjon handler om når vi som brukere skal interagere med datamaskinen. Da er det viktig at brukergrensesnittet lar deg gjøre det du ønsker, og at det er intuitivt å forstå hvordan du skal utføre oppgavene. Å designe god brukerinteraksjon er et viktig felt innen systemutvikling

Feilsøke programmer

En del av programmeringsprosessen går ut på å systematisk feilsøke og forbedre koden. Feilsøking kan gjøres i forbindelse med at man tester programmet.

Funksjoner

En funksjon er et sett av instruksjoner som til sammen utfører en oppgave. I noen programmeringsspråk kalles dette funksjon, i andre kalles det metode. Å lage funksjoner er lurt dersom man skal gjøre oppgaven flere ganger underveis i programmet. Da kan man lage en funksjon, f.eks. for å tegne en trekant, og så hente denne funksjonen hver gang man trenger den.

Løkker

Løkker (eng: loop) er en datastruktur som brukes veldig mye i programmering. Når vi programmerer, så ønsker vi ofte å gjenta identiske instruksjoner i programmet vårt, uten å måtte skrive mye kode om igjen. Da bruker vi løkker istedenfor, og spesifiserer hvor mange ganger instruksjonene skal gjentas. Spesifikasjonen kan være et gitt antall ganger (for-løkker), eller til en betingelse er oppfylt (while-løkker).

Koding

Koding er importert fra det engelske "coding" som i vår sammenheng betyr å uttrykke noe i form av et dataprogram. Programteksten kalles ofte for kode og handler om å utvikle egne programmer ved hjelp av ulike programmeringsspråk. I dette inngår å bruke og forstå grunnleggende prinsipper i programmering, slik som løkker, tester, variabler, funksjoner og enkel brukerinteraksjon.

Kommentere kode

Koden skal kunne leses og forstås av andre, og det er derfor viktig å skrive kommentarer til koden. Disse kommentarene må markeres slik at ikke datamaskinen leser det som en del av programmet.

Modellering

I IT-sammenheng betyr modellering prosessen med å komme fram til en overordnet beskrivelse eller modell av prosessen som utføres av programmet. Modellen omfatter også ofte hvordan programmet brukes. Modelleringen leder til en beskrivelse av hva som skal programmeres. Til dette hører kunnskap om hvordan datamaskiner virker, og hva slags problemer som egner seg for å løses ved programmering. Modellering innebærer også å ta et komplekst problem og dele det, bryte det ned/forenkle det slik at vi kan programmere løsninger på problemet. Modellering av matematiske og naturvitenskapelige fenomener er også en sentral del av hovedområdet. Modellering i denne sammenheng handler om å lage en forenklet framstilling (en modell) av virkeligheten og representere den ved hjelp av en datamaskin. Modellen vil være en forenkling av virkeligheten, og må tilpasses ut ifra hva man skal bruke den til. For eksempel kan man modellere en gass ved å lage et program som kjenner fart og posisjon til alle partiklene, og så hvert sekund regner ut hvordan.

Program

Et program er en oppskrift som sier hva datamaskinen skal gjøre. Programmer skrives i et programmeringsspråk, for eksempel Scratch, Python eller Java, og gir en rekke instruksjoner som forteller hva en datamaskin skal gjøre. Instruksjonene er et sett bestemte kommandoer som settes sammen for å utføre en oppgave eller løse et problem. Et program er skrevet i et programmeringsspråk.

Programmering

Programmering er å lage et program for datamaskinen. Begrepet programmering kan også omfatte prosessen med å strukturere oppgaven som skal løses, og dele den opp i mindre biter som til slutt kan løses ved hjelp av de funksjonene som finnes i et programmeringsspråk. Programmering omfatter også arbeidet med å teste programmet, finne feil og rette dem. Programmering er vanligvis et ledd i en større prosess som

kan kalles *systemutvikling*. Selve prosessen med å skrive programmet kalles i IT-sjargong ofte for *koding*. Det inneholder kartlegging av brukerens behov og design av en overordnet struktur for det som skal lages. Det handler om å lage programkode, det vil si et sett med regler og uttrykk for å styre digitale enheter. I dette inngår prosessen fra å identifisere problemer og utforme mulige løsninger, til å lage kode som kan forstås av en datamaskin, systematisk feilsøke og forbedre denne koden, og dokumentere løsningen på en forståelig måte. Det omfatter alle nivåer fra å forutse og analysere hva et program skal gjøre, til å kjenne igjen mønstre, eksperimentere og evaluere mulige løsninger, og samarbeide med andre.

Programmeringsspråk

Et programmeringsspråk er en mengde kommandoer som får datamaskinen til å utføre gitte instruksjoner. Det finnes mange forskjellige programmeringsspråk, som alle har sine fordeler og ulemper, og det er dermed viktig å velge et programmeringsspråk som egner seg til oppgaven en skal utføre. Vi deler ofte programmeringsspråk inn i blokkbaserte eller tekstbaserte programmeringsspråk, ut ifra hvordan kommandoene er representert. Blokkbaserte programmeringsspråk benytter seg av klosser som man setter sammen til større blokker og programmer, litt som å bygge med LEGO. Tekstbaserte programmeringsspråk består av tekstbaserte kommandoer, og det er viktig å følge korrekt syntaks for å få programmet til å kjøre.

Tester

I et program ønsker vi ofte at noe skal skje dersom visse kriterier er oppfylt. Da bruker vi tester (eng: conditional sentences) til å sjekke betingelsene underveis. I programmering bruker vi tester for å sjekke om en betingelse er oppfylt, dette kaller vi ofte for if-setninger eller if-else-setninger.

En if-setning sjekker om noe er sant eller ikke, for eksempel om rett passord er tastet inn. Andre sammenhenger man kan tenke seg, er hvis du skal betale med bankkortet ditt; da vil programmet først sjekke at du taster inn riktig pin-kode og deretter sjekke om du har nok kroner på kontoen til at kjøpet kan gjennomføres.

Variabler

Variabel er en bokstav eller et ord man kan bruke for å vise til noe annet. Slik kan en variabel ses på som en plassholder for noe annet. En variabel kan defineres som, eller stå for tekst eller verdi. Når en variabel er definert, kan den brukes i koden, og datamaskinen vil da lese variabelen som det den er definert som.