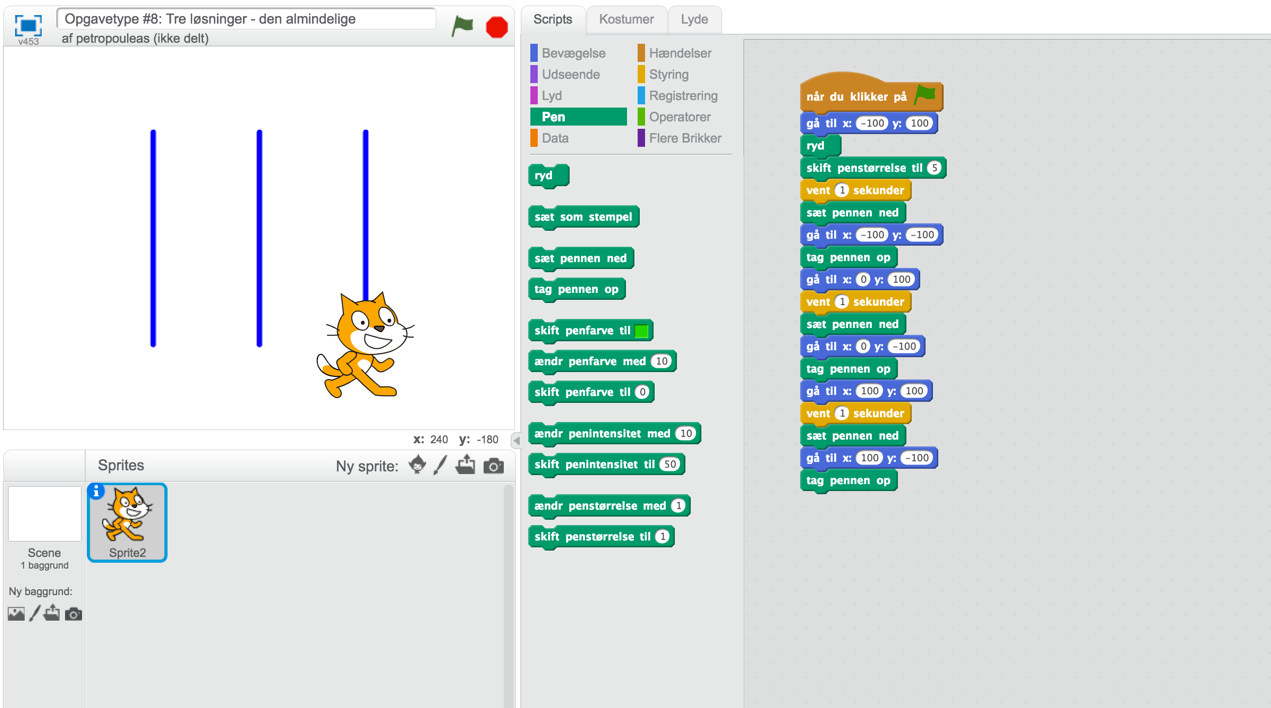
**Opgavetype #8: Tre løsninger**

Elev-handout

Her ser du en almindelig løsning på et program, som får katten til at tegne tre lodrette streger med en pause imellem af 1 sekunds varighed:



**Opgave**

Du skal nu prøve at lave henholdsvis en smart og en vanskelig løsning på   
  
samme problem.

Den smarte løsning forenkler koden mest muligt – tænk fx over, hvordan du kan bruge disse blokke:



Den vanskelige løsning kan eksempelvis være en løsning, hvor du har tænkt en ekstra funktion ind i programmet. Tænk fx over, om du kan lave programmet, så man selv kan bestemme, hvornår stregerne tegnes og hvilken streg, der skal tegnes først.

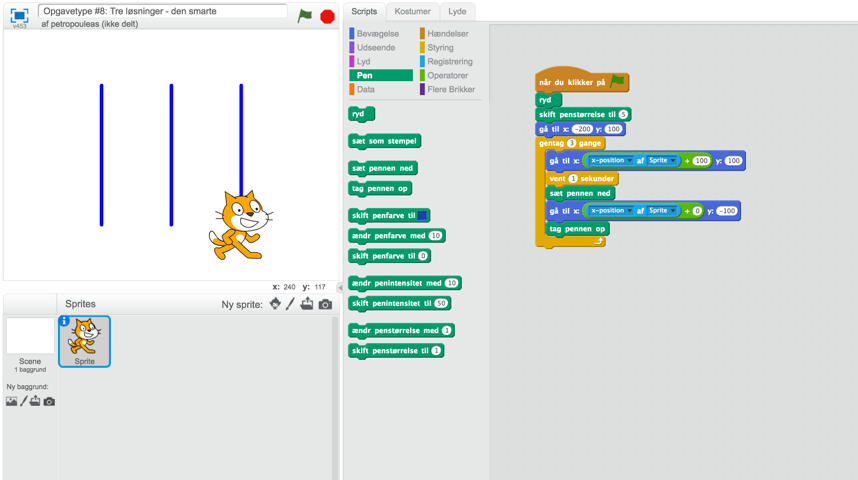
**Til underviseren**

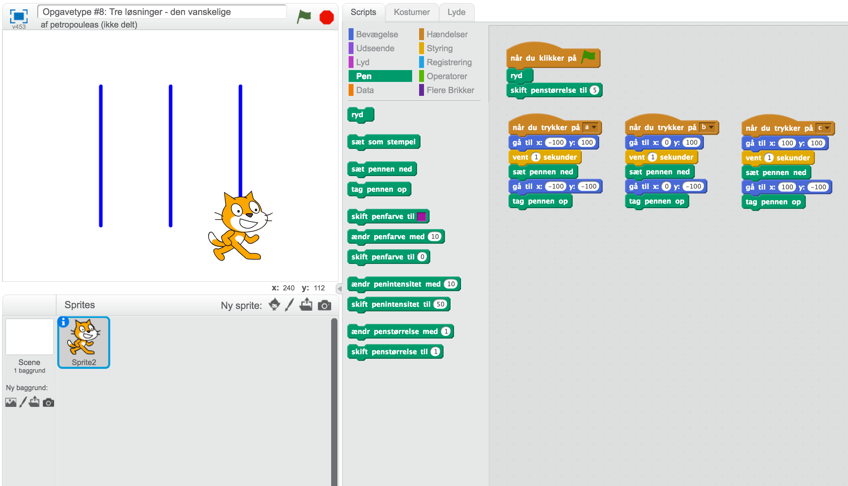
Opgavetype #8: ”Tre løsninger”

**Løsning**

Eksemplet på den smarte løsning benytter en løkke + en matematisk operator til at placere spriten på sin X-position + 100 pixels, hver gang løkken gentages. Derfor er den første placering af spriten inden løkken også 100 pixels længere til venstre, end den er i den almindelige løsning.





Eksemplet her på en vanskelig løsning er ikke så svær programmeringsmæssigt, men tilføjer alligevel 2 nye funktioner. Ved at splitte programmet op i 4 mindre programmer er det muligt at lade brugeren bestemme, hvilken streg der skal tegnes hvornår:



**Hvad eleven bør vide på forhånd**

Eleven skal kunne åbne Scratch, samt vide, hvordan de forskellige kodeblokke er organiseret, hvordan de sættes sammen og fjernes igen. De skal også vide, at de selv kan ændre på tallene i de hvide cirkler, som indgår i nogle af kodeblokkene, og hvor det grønne flag, som aktiverer programmet, er. I den konkrete opgave skal eleven også kende til Scratch’s brug af koordinatsystem til placering af sprites, løkker og matematiske operatorer.

**Beskrivelse af opgavetype #8: Tre løsninger**

Pind (2015) anbefaler i sin bog om åben og undersøgende matematik, at eleverne til hver opgave udarbejder tre løsninger: den almindelige, den vanskelige og den smarte. Det er en fin anbefaling, også når der programmeres, da det er med til at åbne elevernes forståelse for, at der findes flere løsninger på samme problem, og at nogle løsninger er mere “elegante” end andre. I programmering vil den almindelige løsning ofte være en løsning, som løser opgaven, men hvor der ikke er tænkt synderligt over at forenkle koden mest muligt.

Det er der til gengæld i den smarte løsning, hvor koden er forenklet med brug af de muligheder, som eleverne kender til, eksempelvis løkker, indlejrede forgreninger og procedurer. Hvis eleverne allerede har arbejdet med opgavetype #4, er forholdet mellem den almindelige og den smarte løsning ikke ukendt for dem.

Den vanskelige løsning kan eksempelvis være en løsning, hvor eleven har udfordret sig selv ved at inddrage elementer, som er svære (og måske nye) for eleven eller udarbejdet ekstra funktioner i programmet, som ligger ud over den specifikke opgave, som programmet skulle løse. I princippet kan eleverne tage udgangspunkt i enten en smart, en almindelig eller en vanskelig løsning, og derudfra konstruere de øvrige to. I praksis vil det dog give bedst mening at tage udgangspunkt i en almindelig løsning, da det er ønskværdigt, at eleverne lærer at forenkle mest muligt (den smarte løsning) og at de styrkes i at arbejde eksplorativt med nye elementer (den vanskelige løsning), men næppe den modsatte vej.

***Didaktiske principper:*** *#2: Der tages udgangspunkt i et program, der virker. #3: Programmet afprøves, analyseres og modificeres, inden der konstrueres et lignende program. #5: Trinvis forfinelse; programmet udvides og omstruktureres.*

***Computational Thinking:*** *logisk ræsonnement, abstraktion, mønstre og generaliseringer, evaluering, eksperimentering, udformning.*

***SOLO-taksonomi:*** *relationelt/udvidet abstrakt niveau. Eleven ser sammenhænge mellem programmets koder og dets logik og kan omstrukturere og udvide det, så det bliver mere effektivt.*