Oppgave 1b
Hensikten med alle disse testene er å dekke ulike tilfeller for å sikre at funksjonen *isPalindrome* virker under forskjellige forhold. Videre vil jeg si noe om hver enkelt test.

A: 'anna'

- Tester et enkelt palindrom uten mellomrom eller spesialtegn. Dette er en grunnleggende test for funksjonaliteten. Resultatet er ***true*** fordi "anna" er et palindrom (leses likt I begge retninger).

B: 'eva'

- Tester et kort ord som ikke er et palindrom. Den passer på at funksjonen korrekt returnerer ***false*** da et ord ikke er et palindrom.

C: 'mormor ellemeller om rom'

- Selv om den inneholder flere ord og mellomrom, testes det slik at funksjonen kan håndtere mellomrom korrekt og fortsatt returnere ***true*** for et palindrom.

D: 'morfar ellemeller om rom'

- Denne ligner på forrige test, men "morfar" gjør at setningen ikke er et palindrom. Testen sjekker om små endringer bryter mønsteret. Forventet resultat er ***false***.

E: 'Agnes i senga'

- Sjekker en setning med store bokstaver og mellomrom. Testen ser om store bokstaver blir behandlet riktig. Forventet resultat er ***true.***

F: 'Agnes i sengen'

- Ligner på E, men "sengen" gjør setningen til et ikke-palindrom. Testen sjekker nøyaktighet ved små endringer. Forventet resultat er ***false.***

G: 'Anna, Durek er udanna!'

- Inneholder spesialtegn og store bokstaver. Testen ser om funksjonen ignorerer tegn og finner palindromet. Forventet resultat er ***true.***

H: 'Anne, Durek er udanna!'

- Ligner på G, men "Anne" gjør det til et ikke-palindrom. Testen sjekker små forskjeller i ord. Forventet resultat er ***false.***

I: En ranet Unni sier

- Tester en lang og kompleks setning med tegn og store bokstaver. Testen sjekker om funksjonen håndterer lange palindromer. Forventet resultat er ***true.***

Konklusjon: Hver test sjekker forskjellige situasjoner, som mellomrom, tegnsetting, store bokstaver og små variasjoner i teksten.

Oppgave 2b
Koden i denne oppgaven genererer en HTML-side med ordgåter basert på en ordliste fra JavaScript fil-words.js. Hver gåte inneholder to ord som inneholder felles deler. Brukeren kan se et ufullstendig første og andre ord og kan klikke på en ¨?¨knapp slik at han kan se hele ordkombinasjon. Trinnene i denne koden:

1. Opprinnelig 2 filer: HTML – genererer gåtene og words.js – et liste med mange forskjellige ord.
2. Når siden åpnes, kjøres funksjonen `generateWordPuzzles()` automatisk. Målet er å lage 10 ordgåter.
3. Funksjonen velger tilfeldige ord som er lengre enn 6 bokstaver. Ordet deles opp i første del og siste del.
4. Videre søker funksjonen etter et annet ord i listen som også er lengre enn 6 bokstaver og starter med de samme tre bokstavene som det første ordet slutter med.
5. Når slik ord blir funnet, vises gåten slik: første del av det første ordet, etterfulgt av tre streker og tre streker, etterfulgt av siste del av det andre ordet.
6. En knapp legges til ved siden av. Når brukeren trykker på den, vises hele ordkombinasjonen.

Refactoring er en prosess for optimalisering/forbedring av en kode uten å påvirke dens funksjonalitet slik som at det blir lettere å lese og som resultat forstår det. Man kan bruke refactoring i flere tilfeller, blant annet ved å forenkle komplekse funksjoner, omstrukturere kode for å fjerne duplisering, forbedre navn på variabler og funksjoner, eller separere store kodeseksjoner i mindre, mer håndterbare deler.

Fordeler med refactoring:
- Gjør koden mer robust og pålitelig.
- Lett å legge til nye funksjoner uten problemer ved behov.
- Renere kode gjør det enklere for teamet å jobbe sammen.

Ulemper med refactoring:
- Det kan være tidskrevende. Særlig med lang kode.
- Selve prosessen kan føre til en feil om man ikke teste testen godt nok.
- Noen nevner at refactoring ikke gir nye funksjoner eller umiddelbare gevinster, kan det være vanskelig å rettferdiggjøre det overfor ledelsen eller kunden.