Dersom du finner at oppgaveteksten ikke gir tilstrekkelig informasjon, eller at oppgaveteksten er tvetydig, så må du gjøre dine egne forutsetninger. Disse må du i så fall presisere og begrunne. Dette er spesielt viktig dersom du ønsker å svare på en oppgave som ber deg om å basere svaret på en tidligere oppgave som du ikke har svart på.

Legg spesielt merke til hva hver enkelt oppgave ber deg besvare. Husk at begrunnelsen og beskrivelsen i svarene er viktig og derfor bør være tydelig. Bruk gjerne eksempler for å utdype og presisere svarene der du finner det hensiktsmessig.

Tidsangivelsen på oppgavene sier noe om hvor lang tid du bør sette av til hver oppgave og hvor omfattende svaret bør være. Dette er ikke en fast vekting for totalkarakteren fra hver oppgave. Eksamenskarakteren settes etter en helhetlig vurdering av den innleverte besvarelsen etter beskrivelse fra universitets- og høgskolerådet.

Dersom du får knapp tid er det bedre å ha korte beskrivelser eller skisser på punktene du ikke får tid til, fremfor å la punktene være blanke.

Det kreves ikke omfattende drøftinger i forbindelse med figurer o.l., men du bør gi kommentarer, bl.a. hvis du har gjort vesentlige valg.

Oppgave 1 - Krav (0,8t)

a) En konsertbillettbutikk trenger et brukerscenario som viser hvordan den nye billettappen deres skal brukes.
    i) Lag et scenario (også kalt en "brukerreise") som illustrerer brukssituasjonen for applikasjonen når brukeren ønsker å bestille og senere benytte to billetter til en konsert på Blå Grotte i Fredrikstad. Pass på at scenarioet beskriver brukssituasjonen fra brukeren går inn i appen til konserten er over.
    ii) Lag en liste over kravene til applikasjonen som kommer ut fra scenarioet du har lagd.
b) Hvordan skiller ikke-funksjonelle krav seg fra funksjonelle krav? Gi eksempler.
c) Hva er spesielt viktig med hvordan ikke-funksjonelle krav beskrives?

Oppgave 2 - Containerization og Orchestration (0,4t)

a) Forklar forskjellen mellom containerization og orchestration.
b) Versjonskontroll har vært et gjennomgående tema for semesteret. Hvordan er versjonskontroll relevant for både containerization og orchestration?

Oppgave 3 - Avhengigheter (0,4t)

a) Forklar kort hva semantisk versjonering er, og hvordan semantisk versjonering kan gi oss mer stabil programvare.
b) En låsefil i et avhengighetsystem inneholder som oftest en signatur (en hash) av det som ble lastet ned da avhengigheten ble installert. Hva er hensikten med at denne signaturen ligger i låsefilen?

Oppgave 4 - Versjonskontroll (0,4t)

Forklar hva branching og merging i et versjonskontrollsystem som git er, og hvorfor dette er nyttig for programvareutvikling.

Oppgave 5 - Testing av kode (1,0t)

Det nye, norske Unikbiblioteket trenger et nytt system for å håndtere bøkene sine. De skal kun ha ett eksemplar av hver bok i biblioteket sitt. De første kravene til systemet ser slik ut:

a. En låner skal kunne låne en bok som er tilgjengelig for utlån.
- En utlånt bok er ikke lenger tilgjengelig for utlån

b. En låner skal kunne levere tilbake en bok som er utlånt.
c. En bruker skal kunne se om en bok finnes i systemet.
d. En bibliotekar skal kunne legge til en ny bok i systemet.
- Det kan kun være en bok pr. bok-id i systemet.

Dette gir følgende oss følgende enkle grensesnitt:

```
interface Library {
      boolean addBook(String bookId);
      boolean hasBook(String bookId);
      boolean lendBook(String bookId);
      boolean returnBook(String bookId);
}
```

Hver metode gir true / false tilbake avhengig av om operasjonen ble gjennomført eller ikke.

`addBook` gir `true` dersom boken ble lagt til i biblioteket, `false` dersom boken allerede er lagt inn.
`lendBook` gir `true` dersom boken ble registrert som utlånt, `false` dersom en bok ikke finnes eller ikke er tilgjengelig for utlån.
`returnBook` gir `true` dersom boken ble registrert som levert tilbake, `false` dersom en bok ikke finnes eller ikke er utlånt,

`hasBook` gir `true` dersom boken finnes i systemet (uavhengig av om den er lånt ut eller ikke), og gir `false` dersom boken ikke finnes i systemet.

Du trenger ikke tenke på tilgangskontroll eller tilsvarende - biblioteket stoler selvsagt på alle sine lånere, og blir bare glade om noen andre legger til bøker i det.

Skriv nødvendige tester (som pseudokode eller i valgfritt programmeringsspråk) som sørger for at funksjonalitetene og oppførselene som beskrevet ovenfor blir testet. Strukturér og navngi testene slik at de beskriver scenarioene du tester godt. Eksakt syntaks er ikke viktig, og du kan anta at du har `assertTrue`, `assertFalse`, `assertArrayEquals(String[], String[])` og ev. andre asserts (eller en generell `assert` som i Python) tilgjengelig direkte (import av testrammeverk, decorators etc. trenger du ikke tenke på).