IT2-prøve, tysdag 26. nov. 2024

Tid: 09.50 –14.00 (4 timar og 10 minutt) (utsatt tid: 15.00)

Hjelpemiddel og kjelder:
Alle, med unntak av kommunikasjon. Det er IKKJE lov til å spørre om hjelp i ulike forum, i chat, å bruke KI, Copilot eller liknande. Dette blir sett på som juks.

Du skal oppgje alle kjelder der du hentar kode frå på ein tydeleg måte, der det òg tydeleg går fram kva du har henta.

Alle legg mobilen på bordet, slår den på lydlaus, og lar den ligge der i løpet av dagen.

Ved oppstart viser du at du har slått av eller sletta extensions som Copilot, Gemini eller liknande. Felles for desse er at dei skriv/genererer kode for deg. Det kan bli tatt stikkprøvar undervegs.

Vurdering:
Vurderingskriterier finn du til slutt i dette dokumentet. Det er i tillegg oppgitt nokre endå meir konkrete kriterier/krav i sjølve oppgåvene.

Innlevering:
Marker alle mappene/filene med arbeidet ditt og i:

- Windows: Høgreklikk på mappene/filene og velg “Send til komprimert (zippet mappe)”.
- MacOS: Høgreklikk på mappene/filene og velg “Komprimer (x) objekter”.

Filnavn:

Ditt navn IT2 november 2024

- Last opp zip-fila der prøven blei utlevert. Hugs å trykke på “Lever”-knappen.

Oppgave 1: Løkker og valgsetninger

Anbefalt tid: Ca. 40 minutter.

En BMI-kalkulator er et verktøy som gir svar på hvor høy BMI du har. BMI står for Body Mass Index, og er en formel som viser forholdene mellom vekt og høyde. BMI-en din regnes ut på bakgrunn av din vekt og høyde med følgende formel:

vekt/(høyde målt i meter)(høyde målt i meter)*

Hvis du for eksempel er 1,75 meter høy og veier 75 kg, så ser ditt BMI-regnestykke slik ut:

*75 / (1.75 * 1.75) = 24.49*

1.a

Lag et program som regner ut BMI i eksempelet ovenfor.

1.b

Utvid programmet til å la brukeren skrive inn høyden i **cm**, og vekten i **kg**.
Utskriften skal angis med 1 desimal, eksempelvis: **BMI: 24.5**

1.c

Utvid programmet slik at det ikke feiler dersom brukeren skriver inn noe annet enn heltall. Gi i så fall brukeren en beskjed.

1.d

Utvid programmet til å gå i en evig løkke og avsluttes bare dersom bruken skriver inn en tom tekst for høyde. Avslutt programmet med å skrive ut teksten: **Takk for nå.**
Start programmet med å skrive ut teksten: **BMI-kalkulator**

1.e

Utvid programmet til å ta med hvilken kategori BMI-en tilhører. Vi tar bare med tre kategorier:

- 18,4 eller mindre : Undervektig
- 18,5-24,9 : Normalvektig
- 25 eller mer : Overvektig

Teksten kan stå i parentes som i figuren til høyre.

```
BMI-kalkulator
Skriv inn høyden i cm: 182
Skriv inn vekten i kg: 89
BMI: 26.9 (overvektig)
Skriv inn høyden i cm:
Takk for nå.
```

1.f

Er programmet nå 100% feilfritt, eller finnes det situasjoner hvor programmet feiler eller gir meningsløse svar?

1.g

Tegn et digitalt flytskjema for programmet.

Oppgave 2: OOP, parkeringshus

Anbefalt tid: Ca. 90 minutter.

a) Du skal lage et objektorientert program for et parkeringshus med plass til 20 biler. Kravene til systemet er

- Når en bil kjører inn i parkeringshuset skal bilen og tidspunktet registreres og skrives ut.
- Når en bil kjører ut av parkeringshuset skal parkeringstiden og avgiften beregnes og skrives ut. Prisen er kr 1,50 pr minutt.
- Når en bil kjører inn eller ut av parkeringshuset, skal det skrives ut hvor mange ledige plasser det er igjen.
- Det skal være mulig å skrive ut alle parkerte biler i en oppgitt farge.
- Du skal også håndtere spesielle versjoner av kjøretøy. Når en større bil (la oss si en lastebil) skal parkere, koster det 3,0 pr minutt. Ta hensyn til andre faktorar som du mener må legges til i dette tilfellet. Tips: Lag gjerne en ny versjon av programmet for denne delen, slik vi har snakket om i timene.

b) Lag et UML-diagram for løsningen din.

c) Test ut funksjonaliteten i systemet ved å opprette objekter og kalle metoder. Se eksempel neste side.

d) Hvilke feil kan oppstå i et parkeringshussystem som oppfyller kravene beskrevet ovenfor, dersom systemet baserer seg på manuell registrering? Du kan svare på dette spørsmålet som kommentar i koden.

TIPS

Bruk gjerne en ordbok (`dict`) med registreringsnummer som nøkkel og bilobjektet som verdi for parkerte biler. Det vil gjøre det enklere å håndtere utkjøringen, for da kan vi hente og fjerne en bil fra ordboken med `.pop(reg_nr)`.

Fortsettelse neste side.

Eksempel på kjøring ved bruk av vanlige biler (ikke lastebil):

```python
if __name__ == "__main__":
    phus = PHus()

    bil = Bil("AB12345", "Tesla", "rød")
    tid = datetime(2023, 11, 29, 13, 37)
    phus.innkjøring(bil, tid)

    bil = Bil("EK13002", "eGolf", "hvit")
    tid = datetime(2023, 11, 29, 13, 45)
    phus.innkjøring(bil, tid)

    tid = datetime(2023, 11, 29, 14, 41)
    phus.utkjøring("AB12345", tid)

    bil = Bil("TV12345", "Ford", "hvit")
    tid = datetime(2023, 11, 29, 14, 50)
    phus.innkjøring(bil, tid)

    bil = Bil("VD77077", "Ford", "blå")
    tid = datetime(2023, 11, 29, 14, 55)
    phus.innkjøring(bil, tid)

    phus.vis_biler_med_farge("hvit")
```

```
Opprettet parkingshus med 20 plasser.

Opprettet rød Tesla med reg.nr. AB12345
AB12345 kjørte inn 13:37
Antall ledige plasser: 19

Opprettet hvit eGolf med reg.nr. EK13002
EK13002 kjørte inn 13:45
Antall ledige plasser: 18

AB12345 rød Tesla kjørte ut 14:41.
Parkeringstid: 64 min., avgift kr 96.00
Antall ledige plasser: 19

Opprettet hvit Ford med reg.nr. TV12345
TV12345 kjørte inn 14:50
Antall ledige plasser: 18

Opprettet blå Ford med reg.nr. VD77077
VD77077 kjørte inn 14:55
Antall ledige plasser: 17

Parkerte biler som er hvite:
   EK13002 eGolf
   TV12345 Ford
```

Oppgave 3: PyGame, OOP, "KlikkKlikkKlikk"

Anbefalt tid: 90 minutter.

Versjon 1:

Brukeren skal trykke så mange ganger på muspekeren som mulig innen en viss tid. Klikkingen må gjøres innen PyGame-vinduet.

Gi spilleren 10 sekunder på å teste seg selv.

Når spilleren er ferdig, får hen opp resultatet. Det er godt nok at dette skrives til terminalen (vha. print).

Versjon 2:

Du skal utvide versjon 1 slik at spillet fortløpende kan spilles videre når en runde er gjennomført.

Dette skal gjøres på følgende måte: Rett etter at resultatet er skrevet ut vha. print, så starter en ny runde. Indiker at en ny runde har startet ved å skifte bakgrunnsfarge på spillbrettet.

Du får i tillegg en utskrift som sier om du har fått ny bestetid, eller ei.

Når spilleren lukker spillvinduet, så printes bestetiden ut, og dette skrives i tillegg til en tekstfil i samme mappe som spillet ble startet fra.

Versjon 3:

Brukeren skal fortsatt trykke så fort som mulig på muspekeren, men nå teller ikke trykket uten at man treffer et rektangel. Rektangelet flyttar på seg hvert andre (2) sekund, og dukker opp et tilfeldig sted innen "spillbrettet". Ellers samme regler som før.

Tilpass gjerne spillreglene noe for å gjøre spillet kjekkere å spille. Begrunn valgene dine.

LYKKE TIL!

Vurdering

Kompetansemål	**Høy grad av måloppnåelse**
utforske og vurdere alternative løsninger for design og implementering av et program, vurdere og bruke strategier for feilsøking og testing av programkode, generalisere løsninger ved å utvikle og bruke gjenbrukbar programkode, vurdere brukervennligheten i egne og andres programmer og foreslå forbedringer,	Bruker hensiktsmessig mappe og filstruktur. Strukturerer og kommenterer egen programkode på en presis og konsis måte. Beskriver egne og andres algoritmer og gjør rede for virkemåte steg for steg. Vurderer effektiviteten av egne og andres algoritmer, og gjør avveininger i forhold til nytte vs. prestasjon. Bruker algoritmer for å analysere data og svare på problemstillingen. Vurderer ulike måter å strukturere og lagre informasjon, og kan benytte seg av disse etter behov.
Variablar	• Skriver velstrukturert kode med gode variabelnavn og hensiktsmessige datatyper. • Brukt på ein god måte for å styre programflyt og gjere programmet meir dynamisk. • (Tydelege navn som gjer meining)
Valgsetningar	• Brukt på ein god måte for å styre programflyt og gjere programmet meir dynamisk, og handterer unntak ein blir bedt om. • Handterer feil, og gjer tydelege feilmeldingar.
Løkker	• Løkker som dynamisk itererer gjennom data og "gjer ein jobb", eksempelvis summerer, teller opp antal, samanliknar data og liknande. • Blir brukt til å begrense operasjonar ("så lenge som … ").
Funksjonar	• Kan samle kode i funksjonar der det er hensiktsmessig, legg ikkje sentrale deler utanfor desse som gjer at programmet feilar. • Kan bruke funksjonar med parameter, og som kan returnere data. • Kan skrive generelle funksjonar som kan gjenbrukast. • (Tydelege navn som gjer meining)
Kolleksjonar/datastrukturar	• Kan vurdere kva for ein kolleksjon som egnar seg til ein gitt situasjon, og benytte seg av denne på ein god måte for å løyse ulike problemstillingar/oppgåver.

OOP	• Kan planlegge eit program med tydeleg UML-modell, og begrunne vala som er gjort, samt diskutere eventuelle andre løysingar. • Kan lage ei løysing som tek hensyn til problemstillingane i oppgåveteksten, og nyttar seg av ein god struktur mtp. klassar, arv og andre faktorar frå pensumet. • Får samanhengane mellom ulike klassar til å fungere (bok/bøker er ein del av eit bibliotek, ein bil kan parkere i eit parkeringshus osv.).
PyGame	• Kan lage interaktive applikasjonar og spel som tek hensyn til problemstillingane i oppgåveteksten. • Har god struktur i koden (testing på kriterier, oppdatering, teikning m.m. ligg på ”rett stad”). • Nyttar OOP.

Tips

- Løys ein og ein del, sjå gjerne på det i mange tilfeller som å få ei og ei linje med kode til å fungere. Få denne til å køyre før du går vidare. Etterkvart kan me sleppe oss meir laus, og skrive litt meir kode om gongen. Kvifor? Til dømes er det lettare å feilsøke, i mange tilfeller.
- Få grunnfunksjonaliteten på plass først, så kan du be om input, og gjere andre deler som typisk kan by på fleire problem. Eksempelvis kan du late som om du har fått noko input ved å definere ein variabel som du heilt tydeleg styrer i byrjinga av programmet.
- Ha ei testfil der du køyrer mindre deler av koden for å sjekke om til dømes oppsettet av ei løkke er feil, eller om det er andre deler av programmet som feilar.