2020

class Emne:

    def \_\_init\_\_(self, emnekode, studenter, rettere):

        self.\_emnekode = emnekode

        self.\_studenter = studenter

        self.\_rettere = rettere

        self.\_obliger = {}          # {string:Oblig} Ikke gitt i oppgaven.

# 4b) 5 poeng

    def \_skrivMeny(self):

        print ("Behandler emnet " + self.\_emnekode + ". Lovlige kommandoer:")

        print ("O: Ny oblig")

        print ("F: Frist ute, start retting")

        print ("L: Lag eksamensliste")

        print ("A: Avslutt")

    def administrer(self):

        self.\_skrivMeny()

        kommando = input("Gi kommando: ").strip().upper()

        while kommando != "A":

            if kommando == "O":

                self.\_opprettOblig()

            elif kommando =="F":

                self.\_startRetting()

            elif kommando =="L":

                self.\_skrivEksamensListe()

            else:

                print("Ukjent kommando.")

            self.\_skrivMeny()

            kommando = input("Gi kommando: ").strip().upper()

# 4g) 15 poeng - poeng gis samlet for alle tre metoder.

    def \_opprettOblig(self):

        obligId = "oblig" + str(len(self.\_obliger)+1)

        frist = input("Oppgi frist for " + obligId + "paa formen aammdd: ")

        self.\_obliger[obligId] = Oblig(obligId, frist)

    def \_startRetting(self):

        dagensDato = input("Oppgi dagens dato pa formen aammdd: ")

        for obligId in self.\_obliger:       # Noekkel er streng obligId

            oblig = self.\_obliger[obligId]  # Referanse til Oblig-objekt

            if oblig.klarForRetting(dagensDato):

                besvarelser = oblig.hentBesvarelser()

                resultater = oblig.fordelRetting(besvarelser, self.\_rettere)

                for sBruker in resultater:

                    stud = self.\_studenter[sBruker]

                    res = resultater[sBruker]

                    stud.registrer(obligId, res)

    def \_skrivEksamensListe(self):

        eksamensliste = []

        for sBruker in self.\_studenter:

            stud = self.\_studenter[sBruker]

            if stud.altGodkjent(len(self.\_obliger)):

                eksamensliste.append(sBruker)

        print("Eksamensliste for emnet "+self.\_emnekode+":")

        for sBruker in eksamensliste:

            print(sBruker)

#4c) 10 poeng

class Student:

    def \_\_init\_\_(self, brukernavn, fulltNavn):

        self.\_bruker = brukernavn

        self.\_navn = fulltNavn

        self.\_obliger = {}

    def registrer(self, obligId, resultat):

        self.\_obliger[obligId] = resultat

    def altGodkjent(self, antObliger):

        if antObliger > len(self.\_obliger): # Noen obliger ikke levert/ registrert

            return False

        for obl in self.\_obliger:

            if self.\_obliger[obl] != 1:     # Oblig ikke godkjent

                return False

        return True

# 4d) 2 poeng

class Retter:

    def \_\_init\_\_(self, brukernavn):

        self.\_bruker = brukernavn

    def vurder(self, filnavn):

        return 1

# 4e) 8 poeng

class Oblig:                                # NB: Ikke besvarelser

    def \_\_init\_\_(self, obligId, frist):

        self.\_obligId = obligId

        self.\_frist = frist

        self.\_rettet = False

    def klarForRetting(self, dagensDato):

        return dagensDato > self.\_frist and not self.\_rettet

# Filnavn genereres i metoden ut fra entydig obligId

    def hentBesvarelser(self):

        filnavn = self.\_obligId + ".txt"

        alleBesvarelserFil = open(filnavn, "r") # 1 linje per student for denne oblig

        besvarelser = {}                        # {student:filnavn med besvarelse}

        for linje in alleBesvarelserFil:

            bData = linje.split()

            if len(bData) > 1:                  # kan mangle besvarelse

                besvarelser[bData[0]] = bData[1]

        alleBesvarelserFil.close()

        return besvarelser                      # alle studenters oblig-filer (1 oblig)

# 4f) 5 poeng

    def fordelRetting(self, besvarelser, rettere):

        resultater = {}

        antR = len(rettere)

        rNr = 0

        for sBruker in besvarelser:

            retter = rettere[rNr]

            res = retter.vurder(besvarelser[sBruker])

            resultater[sBruker] = res

            rNr += 1

            if rNr == antR:

                rNr = 0

            # Alternativt: rNr = (rNr + 1) % antR

        self.\_rettet = True

        return resultater

# Hovedprogram er ikke gitt i oppgaven og skulle ikke skrives.

# Eksempel, ikke komplett testprogram.

def hovedprogram():

    # emnekode = input("Hvilket emne skal administreres? ")

    studenter = {}

    for i in range(1, 5):   # Genererer data med "kjedelige" navn

        brukernavn = "s"+str(i)

        ny = Student(brukernavn, "Fulltnavn student: " + brukernavn)

        studenter[brukernavn] = ny

    rettere = []

    for i in range(1,3):   # Genererer data med "kjedelige" navn

        nyR = Retter("R"+str(i))

        rettere.append(nyR)

    emne = Emne("IN1000", studenter, rettere)

    emne.administrer()

    print("Programmet avsluttes")

hovedprogram()

Universitetet i Ruritania skal digitalisere sin håndtering av studentenes obligatoriske oppgaver  
(«obliger»), og du skal lage deler av en pilot med et objektorientert design basert på innkapsling, dvs.  
instansvariable og non-public metoder (navn starter med \_) skal kun benyttes i den klassen de er  
definert. Det er i mange av deloppgavene nødvendig å bruke metoder fra andre klasser/deloppgaver. Du kan bruke disse selv om du ikke har skrevet dem.Emnene har ulike antall obligatoriske oppgaver som kreves godkjent, og hver oblig i et emne må  
registreres manuelt hvert semester. En student må ha alle oppgaver gitt i emnet godkjent for å få gå  
opp til eksamen, men i denne piloten foretas det ikke kontroll av om studenten har fått godkjent  
forrige oblig før godkjenning av besvarelse for neste oblig. Foruten obliger har et emne registrerte  
studenter og ansatte rettere. Både rettere og studenter har unike brukernavn.  
Innleveringen av besvarelser for en oblig foregår i et annet system som du ikke skal lage. Dette  
systemet lager en fil for hver oblig, med et entydig navn etterfulgt av «.txt», for eksempel oblig1.txt.  
Filen inneholder en linje per student på følgende format (eksempel med to linjer):  
student1 obl1student1.txtstudent2I eksempelet er student1 et entydig brukernavn og obl1student1.txt er navnet på filen der student1s  
besvarelse ligger. student2 i eksempelet har ikke levert noen besvarelse på oblig 1.  
Hver besvarelse skal vurderes av en retter for emnet. Når fristen for en obligatorisk oppgave er  
utløpt, fordeles besvarelser på retterne, som vurderer en og en til godkjent (1) eller underkjent (0).  
Piloten har funksjonalitet for hente ut en eksamensliste med brukernavn på alle studenter som er  
kvalifisert for eksamen – dvs har godkjent alle registrerte obliger i emnet.  
Oppgave 4a) 5 poeng  
Definer en klasse Emne (inkludert konstruktør) som holder orden på og administrerer obliger, rettere  
og studenter i et emne. Konstruktørens parametere gir verdier til emnekode (streng), registrerte  
studenter (ordbok med Student-objekter) og rettere (liste med Retter-objekter).  
Oppgave 4b) 5 poeng  
Skriv metoden administrer i klassen Emne. Metoden skriver ut emnekoden for emnet og en meny på  
terminalen før den ber om kommando. Den tar imot følgende lovlige kommandoer fra en bruker:  
O: Ny oblig  
F: Frist ute, start retting  
L: Lag eksamensliste  
A: Avslutt  
Annen input skal gi feilmelding og nytt forsøk. En kommando bør gjenkjennes selv om det er blanke  
foran eller bak, og uansett om brukeren skriver liten eller stor bokstav (metodene strip og upper er  
nyttige her). For å utføre kommandoene skal metoden kalle på følgende non-public metoder som  
også ligger i klassen Emne:  
O -> \_opprettObligF -> \_startRettingL -> \_skrivEksamensListeDisse metodene skal skrives i senere deloppgaver. De opererer kun på instansvariabler i klassen, og  
har ingen parametere (annet enn self) eller returverdier.  
Oppgave 4c) 10 poeng  
Skriv en klasse Student med instansvariabler for brukernavn og fullt navn (begge får verdi fra  
parametere til konstruktøren) og en ordbok som skal ta vare på resultater av rettinger, der obligId er  
nøkkel og resultatet verdien. Klassen har følgende metoder i tillegg til konstruktør:  
registrer med parametere oblig (en streng som identifiserer en oblig entydig) og resultat (et heltall  
som angir om en besvarelse er vurdert som godkjent). Metoden registrerer hvilket resultat en  
student har fått på en oblig.  
altGodkjent med parameter antObliger som angir antall obliger registrert i emnet. Metoden  
returnerer en boolsk verdi: True hvis studenten har fått alle obliger godkjent, False om en eller flere  
obliger mangler eller har et annet resultat enn godkjent (kodet som heltallet 1).  
Oppgave 4d) 2 poeng  
Skriv klassen Retter med instansvariabel retterens brukernavn (parameter til konstruktøren) og  
metoden:  
vurder med en parameter som angir et filnavn for en besvarelse. I denne foreløpige utgaven av  
piloten ignorerer programmet ditt filen med besvarelsen, og returnerer alltid 1 (godkjent).  
Oppgave 4e) 8 poeng  
Skriv en klasse Oblig med tre instansvariabler: Entydig id for obligen, en leveringsfrist på formen  
ååmmdd (f. eks. 190906 for 6. september 2019) og om obligen er rettet eller ikke. De to første  
initialiseres med parametere til konstruktøren. Skriv følgende metoder i klassen Oblig:  
klarForRetting med parameter som angir dagens dato. Metoden returnerer True hvis fristen er før  
dagens dato og besvarelsene for obligen ikke allerede er rettet – ellers False.  
hentBesvarelser leser en fil med oversikt over studenters besvarelser (navn og format på denne filen  
er beskrevet i oppgave-innledningen) og returnerer en ordbok der en students brukernavn er nøkkel  
og filnavn med studentens besvarelse er verdi.  
Oppgave 4f) 5 poeng  
Utvid klassen Oblig med en ny metode  
fordelRetting tar som parametere en ordbok med studenters besvarelser og en liste med rettere, og  
lar retterne vurdere én og én besvarelse (bruk metoden vurder i klassen Retter) slik at alle retterne  
ideelt sett vurderer like mange. Altså skal ingen retter vurdere mer enn 1 besvarelse mer enn de  
andre. Resultatene av retternes vurderinger samles i en ordbok der nøkkelverdiene er studentenes  
brukernavn. Metoden markerer obligen som rettet og returnerer ordboken med resultater.  
Oppgave 4g) 15 poeng  
Skriv følgende non-public metoder i klassen Emne:  
\_opprettOblig genererer et unikt oblignavn, og ber bruker på terminalen om å oppgi frist på formen  
ååmmdd. Deretter legges en ny oblig til emnet.  
\_startRetting ber om dagens dato og sjekker om noen av obligene i emnet har en utløpt frist uten at  
besvarelsene er rettet. I så fall henter den alle besvarelser for obligen, fordeler dem på rettere og  
registrerer resultatet av rettingen i Student-objekter (på riktig oblig) for studenter som har levert.  
\_skrivEksamensListe bygger opp en liste med brukernavn for alle studenter som har fått godkjent  
alle obliger registrert for emnet, og skriver til sist ut denne på terminal med en passende overskrift.

**Oppgave 5**

Karl vil kunne sikre seg at han er forskånt for å måtte lese et bestemt fy-ord og dets synomer.

Skriv en funksjon **sjekk\_om\_fyord(setning, fyord, synonym\_liste)**.

Funksjonen tar som argument en setning (en streng bestående av flere ord med mellomrom mellom) , et fyord

(en streng bestående av ett enkelt ord uten noen mellomrom) og en samling av ulike synonymer (en liste av

lister, hvor hver indre liste består av streng-verdier som er enkeltord). Funksjonen skal sjekke om minst ett av

ordene i setningen er et synonym med det oppgitte fyordet (ifølge den oppgitte samlingen av synonymer) eller

om fyordet i seg selv finnes i setningen. Dersom fyordet eller et synonym av fyordet finnes i setningen skal

funksjonen returnere True. Hvis ikke skal funksjonen returnere False.

Altså skal for eksempel kallet *sjekk\_om\_fyord("spis masse godsaker", "snop", [["saft","lemonade"],*

*["snacks","snop","godsaker"],["mye","masse"]])* returnere True, mens kallet *sjekk\_om\_fyord("spis masse*

*godsaker", "lemonade", [["saft","lemonade"],["snacks","snop","godsaker"],["mye","masse"]])* skal returnere

False.

#5a

def sjekk\_om\_fyord(setning, fyord, synonym\_liste):

    biter = setning.split()

    for bit in biter:

        for synonymer in synonym\_liste:

            if bit in synonymer and fyord in synonymer:

                return True

    return fyord in biter

assert sjekk\_om\_fyord("spis masse godsaker", "snop", [["saft","lemonade"],["snacks","snop","godsaker"],["mye","masse"]]) == True

assert sjekk\_om\_fyord("spis masse godsaker", "godsaker", [["saft","lemonade"],["snacks","snop","godsaker"],["mye","masse"]]) == True

assert sjekk\_om\_fyord("spis masse godsaker", "godsaker", []) == True

assert sjekk\_om\_fyord("spis masse godsaker", "lemonade", [["saft","lemonade"],["snacks","snop","godsaker"],["mye","masse"]]) == False

assert sjekk\_om\_fyord("spis masse godsaker", "agurk", [["mye","masse"], ["spis","gomle"],["snacks","snop","godsaker"]]) == False

**Oppgave 3a**

Knut vil beregne pris inkludert frakt når han skal handle på nettet.

Skriv en funksjon **pris\_inkl\_frakt(varepris)**.

Dersom varepris er over 1000 kroner legges det ikke på noe kostnad til frakt og funksjonen skal returnere

samme varepris som den fikk inn som argument. Dersom varepris er mellom 500 og 1000 kroner (fra og med

500, til og med 1000) skal det legges på frakt, slik at funksjonen returnerer en pris som er 50 kroner høyere enn

varepris. Dersom varepris er under 500 kroner skal det både legges på frakt og ekstragebyr, slik at funksjonen

skal returnere en pris som er 80 kroner høyere enn varepris.

Med andre ord skal f.eks. kallet pris\_inkl\_frakt(300) evaluere til 380, pris\_inkl\_frakt(600) evaluere til 650, og

pris\_inkl\_frakt(1300) evaluere til 1300.

#3a

def pris\_inkl\_frakt(varepris):

    if varepris > 1000:

        return varepris

    elif varepris >= 500:

        return varepris + 50

    else:

        return varepris + 80

assert pris\_inkl\_frakt(300) == 380

assert pris\_inkl\_frakt(600) == 650

assert pris\_inkl\_frakt(1300) == 1300

**Oppgave 3b**

Per vil lage seg en handleliste og raskt kunne fjerne utsolgte varer fra denne lista.

Skriv en funksjon **fjern\_utsolgte(handleliste, utsolgte).**

Funksjonen tar som argumenter en liste med navn på varer (handleliste: liste av streng-verdier) og en liste

over utsolgte varer (utsolgte: liste av streng-verdier). Funksjonen skal returnere en ny liste som inneholde de

ønskede varene som ikke er utsolgte, dvs en ny liste som inneholder alle streng-verdier som finnes i listen

handleliste, men ikke finnes i listen utsolgte. I tillegg skal funksjonen skrive til terminalen navnet på hver vare

fra handlelisten som er utsolgt.

Altså skal for eksempel kallet fjern\_utsolgte( ["melk", "brus", "pasta"], ["kanel","brus"] ) evaluere til listen

["melk", "pasta"], samtidig som teksten brus skrives til terminalen.

#3b

def fjern\_utsolgte(handleliste, utsolgte):

    nyliste = []

    for vare in handleliste:

        if not vare in utsolgte:

            nyliste.append(vare)

        else:

            print(vare)

    return nyliste

assert fjern\_utsolgte( ["melk", "brus", "pasta"], ["kanel","brus"]) == ["melk", "pasta"]

**Oppgave 3c**

Lise skal reise som backpacker gjennom flere land i asia og vil finne ut hvilke vaksiner hun må ta før reisen.

Skriv en funksjon **samlet\_vaksinasjon(krav\_hvert\_land).**

Funksjonen tar som argument en liste av lister hvor hver indre liste er hvilke vaksiner som trengs for et av

landene hun skal reise til. Funksjonen skal returnere en samlet liste over hvilke vaksiner som trengs, dvs en

liste av streng-verdier som inkluderer alle vaksinene funnet i et av landene. Dersom samme vaksine finnes i

lista for mange ulike land, skal den bare være med én gang i den returnerte lista.

Altså skal for eksempel kallet samlet\_vaksinasjon([["difteri","tyfoid"], ["hepatit","difteri"]]) evaluere til en liste

['difteri', 'tyfoid', 'hepatit'] (rekkefølgen av vaksinene i den returnerte lista er uviktig).

#3c

def samlet\_vaksinasjon(krav\_hvert\_land):

    vaksiner = []

    for krav in krav\_hvert\_land:

        for vaksine in krav:

            if not vaksine in vaksiner:

                vaksiner.append(vaksine)

    return vaksiner

assert samlet\_vaksinasjon([["difteri","tyfoid"], ["hepatit","difteri"]]) == ['difteri', 'tyfoid', 'hepatit']

**Oppgave 3d**

Koden under definerer en variabel setning og lager varianter av denne setningen hvor først ordet "en" er fjernet

og deretter også ordet "skal" er fjernet. Det er store likheter mellom kodelinjene som lager setning uten ordet

"en" og kodelinjene som lager setning uten ordet "skal". Skriv en alternativ versjon av programmet som unngår

denne redundansen ved å definere en funksjon forkort\_setning og som kaller denne funksjonen to ganger for å

lage setning uten henholdsvis ordene "en" og "skal". Skriv det fulle resulterende programmet, inkludert

definisjon av funksjonen (og dens parametre), kall på funksjonen (med argumenter) og andre kodelinjer som

trengs for å oppnå samme funksjonalitet som den opprinnelige koden under.

setning = "en krabbe skal en dag ut av skallet "

#fjerner alle ord "en":

setning\_v2 = ""

for ord in setning.split():

if not ord=="en":

setning\_v2 = setning\_v2 + ord + " "

#fjerner alle ord "skal":

setning\_v3 = ""

for ord in setning\_v2.split():

if not ord=="skal":

setning\_v3 = setning\_v3 + ord + " "

print(setning\_v3) #krabbe dag ut av skallet

#3d

def forkort\_setning(setning, fjern):

    ny\_setning = ""

    for ord in setning.split():

        if not ord==fjern:

            ny\_setning = ny\_setning + ord + " "

    return ny\_setning

setning = "en krabbe skal en dag ut av skallet "

setning\_v2 = forkort\_setning(setning, "en")

setning\_v3 = forkort\_setning(setning\_v2, "skal")

assert setning\_v3 == "krabbe dag ut av skallet "