Datortentamen 2024-01-09

TDDE24 Funktionell och imperativ programmering del 2

Allmän information

Tentan består av totalt **6 uppgifter** som kan ge maximalt 5 poäng.

Poänggränserna sätts inte i förväg för denna tenta, delvis för att tentans sammansättning har ändrats jämfört med tidigare år. Exempelvis har denna tenta inte någon uppgift med krav på rekursiv lösningsmodell, och i flera uppgifter har vi hjälpt till med extra testfall, extra förklaringar och tips, och mer detaljerade genomgångar av tänkta algoritmer. (Det är alltid examinatorns uppgift att avgöra *vilka lösningar som uppnår ett visst betyg*, och poänggränser är bara ett av många möjliga hjälpmedel.)

Vad som är lätt eller svårt skiljer sig från person till person. Därför bör man inte förutsätta att de första uppgifterna är enklast, utan snabbt läsa genom alla uppgifter för att kunna prioritera arbetet. Vissa uppgifter kan också ge delpoäng för att man löser specifika delar, så om en uppgift verkar svår i sin helhet kanske du ändå vill arbeta med en del av den!

Tillåtna hjälpmedel och verktyg

Följande hjälpmedel och verktyg är tillåtna / tillgängliga:

- **Penna** för att skissa lösningar på papper. (Tomma papper ska finnas tillgängliga.)
- **Python 3.11**, som vi har använt under senaste kursomgången. Detta ska finnas som kommandot python3.11 på tentadatorerna.

  I terminalfönstret *bör* det också gå att använda Python 3.11 genom att skriva python eller python3, men dessa "alias" fungerar inte nödvändigtvis om du kör program inuti en editor som vscode. Om du får fel version av Python inuti en editor har våra nya svarsmallar en testfunktion som ska tala om det automatiskt vid testkörningen, och det är då upp till dig själv att lösa problemet: Peka ut korrekt version av Python för editorn, eller kör helt enkelt allt med python3.11 från kommandoraden.
- **Ett antal editorer**, inklusive vscode (kommandot `code` på kommandoraden) och gvim. Fler editorer kan finnas tillgängliga från menyer eller från kommandoradsprompten.

  Editorerna har inte alla plugins installerade. Meningen är att man ska få tillgång till de grundläggande menyer och tangentbordsbindningar som man är van vid, inte att man ska ha en fullständig integrerad utvecklingsmiljö. Exempelvis kan man behöva köra program från kommandoraden, och verktyg som debuggers kanske inte fungerar.
- Standardiserade **systemprogram** i kommandoradsprompten eller menyerna.
- **Websidor** med biblioteks- och språkreferenser för Python. Enligt de generella instruktionerna för datortentor ska dessa finnas tillgängliga via en "Web Access"-ikon på skrivbordet (vi som skapar tentan kan inte själva se tentamiljön). Vid eventuella problem, räck upp handen och be tentavakterna tillkalla teknisk jour.

Otillåtna hjälpmedel; fusk och vilseledande

Otillåtna hjälpmedel inkluderar alla former av elektronisk utrustning utöver tentadatorerna, samt böcker och anteckningar.

Under datortentan arbetar du i en begränsad och övervakad datormiljö där fullständiga utvecklingsmiljöer som PyCharm inte är tillåtna, och där du har begränsad tillgång till nätet.

Utöver detta gäller följande:

- Man får **inte kopiera text eller lösningar direkt från andra källor.** Man måste skriva koden på egen hand och **förstå och beskriva** vad den gör.
- Man får **inte kommunicera med andra under tentan**, vare sig för att diskutera uppgifterna eller för andra syften (utom för att ställa frågor till tentavakter, så klart).
- Man får **inte göra tentasvar eller relaterad information tillgängliga för andra** på något sätt under tentans gång. Alla uppgifter ska genomföras helt individuellt.

Vi påminner om att vi är **skyldiga att anmäla** möjligt fusk eller "försök till vilseledande" till disciplinnämnden, utan att själva försöka reda ut om det faktiskt var fusk.

Svarsmallar

För varje uppgift i tentan *skickar vi med* en “svarsmall", med filnamn från ex1.py till ex6.py. **Mallen ska alltid användas som grund till din inlämnade uppgift** och du ska varken döpa om den eller skapa egna filer som lämnas in. **Kopiera** mallfilen från den skrivskyddade katalogen `given_files` till den katalog där du arbetar med uppgifterna, t.ex. `Desktop`. Skriv din kod överst i filen och dina eventuella tester inuti den fördefinierade funktionen `run_tests()` – det underlättar vid rättningen. Provkör med t.ex. “`python3.11 ex1.py`" från kommandoraden.

Mallen hjälper till med detta:

- Testar att man kör rätt version av Python, så att man inte får underliga buggar på grund av fel Pythonversion. Talar annars om vilken version som används.
- Inkluderar “tomma" funktionsdeklarationer för de funktioner du behöver skriva, så du slipper klippa och klistra och inte riskerar att skriva fel. Funktionerna består bara av kommandot `pass`, som du då ska *ersätta* med din egen kod för funktionen. Se upp så du inte får dubbla deklarationer, både vår medskickade och en egen!

  Du får givetvis skapa egna hjälpfunktioner som tillägg.
- Inkluderar eventuella assert-tester som vi redan har angivit i uppgiften, så du slipper klippa och klistra. (Du kan ändå behöva skapa egna tester, för din egen skull.)
- Ser till att testerna bara körs om man *kör* filen, inte om man *importerar* den (så som vi gör vid rättningen). Det är därför **även dina egna tester** ska vara inuti run_tests().

Mallen kan se ut *ungefär* så här:

```
# Här i början lägger du din egen kod för uppgift 1.

def the_function_you_should_write(seq: list):
    pass

def check_python_version():
    ...färdig kod som kollar att du kör rätt version av Python...

def run_tests():
    # De här testerna står uttryckligen som assertions på tentan.
    print("Kör uppgiftens tester...")
    assert the_function_you_should_write(10) == 42

    # Här lägger du dina egna tester.  Du kan till exempel skapa egna
    # assertions, eller lägga till andra tester såsom enkla utskrifter
    # av resultatet av en körning.
    print("Kör egna tester...")

    # Här kan du lägga tester där du inte vet korrekta svar men
    # ändå kan skriva ut resultatet.  Kanske det kraschar, kanske
    # det är uppenbart fel...
    print("Kör utskriftstester...")
    print("Resultat 1:", the_function_you_should_write(4711))

    print("Har kört alla tester")

if __name__ == '__main__':
    check_python_version()
    run_tests()
```

Under tentan: Lämna in uppgifter

Med **tentaklienten** (som du har fått information om i förväg och som även beskrivs i ett dokument som ska finnas tillgänglig i `given_files`) skickar du in dina svar, med en separat inlämning per uppgift. Svaren måste lämnas in innan tentans slut, då tentasystemet automatiskt stängs!

När du gör en inlämning i tentaklienten anger du också *vilken* uppgift du lämnar in (nummer 1–6). I uppdelade uppgifter lämnas del (a) och del (b) in på samma gång, *i samma fil*. Var försiktig så att du anger rätt uppgiftsnummer!

Det går bra att skicka in en lösning på samma uppgift flera gånger, och den nya inlämningen ersätter då alltid den tidigare. Utnyttja det: **Testa att lämna in någon lösning tidigt**, så du garanterat vet hur det fungerar, och **riskera inte att missa sluttid / deadline**, utan lämna in din nuvarande minst 5 minuter före sluttiden även om du tror du kan vilja utnyttja resten av tiden för att polera svaret mera.

Det finns en meddelandelogg i klienten, men såvitt vi vet kan du inte se exakt vilka filer du har bifogat. Om du är osäker på vad du har skickat in kan du skicka in rätt fil en gång till.

Precis som vid vanliga tentor kommer inga svar att granskas förrän efter tentans slut.

Under tentan: Ställa frågor

Om du har **tekniska problem** (kan inte logga in, har problem med tangentbordet, kan inte skicka in svaret på en fråga ...) kontaktar du tentavakterna som vid behov kan kontakta teknisk jour.

Frågor om tentauppgifterna ställs istället via tentaklienten. Frågorna går då till examinatorn. Tentavakter och teknisk jour kan *inte* svara på tentafrågor.

Spara inte frågorna till slutet. På en vanlig tenta går man genom alla uppgifter så snart man får tillgång till dem, så man kan ställa alla eventuella frågor till examinatorn vid ett eller två korta besök i tentasalen. Under denna datortenta kan du visserligen skicka frågor när som helst, men även här kommer frågorna att besvaras “då och då" och **det kan ta någon timme att få svar** – inte minst för att det kan komma många frågor på samma gång.

Ofta kommer många frågor på slutet, så skickar du frågor för sent kan det hända att du inte hinner få svar i tid för att avsluta din lösning innan tentans slut!

Läs alltså genom uppgifterna i början och ställ frågor i god tid!

Under tentan: Informationsutskick

Information som är intressant för flera tentander kan skickas ut via tentaklienten under tentans gång. Detta brukar oftast handla om påminnelser om sådant som redan står i instruktionerna, men som några studenter har missat eller missförstått.

Utskick och svar på frågor syns bara i en meddelandelogg i tentaklienten. Det kommer inga notifikationer eller popup-meddelanden.

Håll alltså koll på tentaklienten och dess meddelandesystem under tentan. Klienten ger INTE automatiska notifikationer om meddelanden kommer!

Viktiga tips om testning – missa inte poäng i onödan

Utöver rättningskriterierna (nästa sida) vill vi starkt uppmana er att tänka på detta:

- Vi ger ofta flera testfall i varje uppgift. De som är skrivna direkt i `assert`-form finns normalt också med i svarsmallen. Det kan också finnas indirekta tips som del av den löpande texten, vilket normalt *inte* läggs med i svarsmallen men kan fungera som inspiration till fler testfall. **Testa allt du kan och skapa egna variationer!** Ofta hittar man fel som faktiskt är lätta att korrigera.
- De testfall vi ger är bara exempel och täcker definitivt inte allt – man måste också utgå från beskrivningen i texten. **Skapa egna tester** som täcker fler fall!
- Det går *delvis* att testa genom att **köra implementationen med många olika indata** även om du inte vet vad korrekt svar ska bli (`print` istället för `assert`). Ibland fungerar loopar – testkör t.ex. med värden från 0 till 100! Det är inte ovanligt att koden kraschar direkt, eller skriver ut uppenbart felaktiga svar. Då har du hittat ett fel, utan att veta exakt vad det korrekta svaret skulle vara!
- Tänk på att **testa specialfall** som *tomma listor*, *tomma tupler*, *negativa tal*, med mera. Testa också *långa* listor eller *djupt nästlade* listor.
- **Läs exakt vad som står**! En *godtycklig lista* är precis vilken lista som helst, så testa med olika listor, även med sådana där elementen består av nästlade listor eller tupler eller andra datastrukturer. En *sekvens* behöver inte nödvändigtvis vara en lista, så testa även med tupler och kanske en sträng eller två. Ska det fungera för heltal $n \geq 0$ ska det också fungera för $n = 0$, så testa det. Ska funktionen *ta en sekvens och returnera en lista* får den inte returnera en tupel, även om inputsekvensen råkade vara en tupel.
- Tänk på att man ofta ska **klara godtyckliga indata**. Även om de angivna testfallen bara är heltal, kanske det står att man ska klara godtyckliga listor, och alltså vilka element som helst. **Testa då detta!** Fastna inte heller i att ett exempel bara råkar ange *positiva* tal, eller att en lista råkar vara *sorterad*. Återigen, läs vad uppgiften ska klara och skapa egna exempel som testar varierande input.

Att testa är viktigt!

Vi ser ofta *många* fel som väldigt enkelt kunde ha upptäckts och fixats om man bara *testade* sin lösning lite mer. Vad är bäst, att få 4 poäng på 4 uppgifter (16 totalt) eller att hinna med en uppgift till men få 2 poäng per uppgift på grund av bristande testning (10 totalt)?

Men testning fångar inte allt!

Grunden i uppgifterna är alltid att *läsa* och *förstå*. Testerna är inte ett facit och koden kan vara felaktig även om den klarar alla tester.

Egna tester rekommenderas men är ej obligatoriska!

Tester får lämnas in, men det är inte nödvändigt. De påverkar inte poängen.

Rättningskriterier

Brott mot följande allmänna kriterier kan resultera i poängavdrag.

- Lösningen ska givetvis vara **körbar**. Testa alltid **precis innan inlämning** så att din sista finputsning eller dina sista kommentarer inte resulterade i felaktig kod och så att koden inte kraschar när filen importeras vid granskning! Poängavdrag ges vid icke körbar kod. (Misslyckade tester och assertions är OK *om* de ligger i `run_tests()`, eftersom vi inte kör den funktionen vid vår betygsättning.)
- Lösningen ska följa alla de **specifika regler och villkor** som står i uppgiften.

  Den ska också **fungera exakt som i körexemplen** i uppgiften, om inte texten indikerar något annat – men tänk på att koden kan vara felaktig trots att körexemplen fungerar! Lösningen ska vara **generell** och ska fungera för *alla* indata som följer uppställningen i uppgiften. Att en lösning enbart fungerar för listor med begränsad längd eller för vissa storlekar på indata är exempel på signifikanta fel.
- Funktioner och källkodsfiler ska ha **exakt samma namn** som anges i uppgiften. Vi hjälper numera till med detta genom svarsmallarna.
- Man ska kunna köra funktioner **flera gånger** med olika indata och korrekt resultat, utan att ladda om koden däremellan. Se upp med olika former av globalt tillstånd / globala variabler. Se upp med defaultargument och modifiera dem aldrig. Testa själv att köra många testfall i rad.
- Kod ska vara **lättförståelig**. Det innebär t.ex. att egna namn (på parametrar, variabler med mera) ska vara **beskrivande** och följa namnstandarden. Det innebär också att lösningen ska vara **välstrukturerad** och tillräckligt **väldokumenterad** för att en granskare **enkelt** ska förstå hur lösningen fungerar och varför den ser ut som den gör.

  Vi har **inget generellt krav på *docstrings***, men någon form av kommentarer kan som sagt behövas för förståelsen.
- Den implementerade lösningen ska kunna köras inom **rimliga tidsramar**. Till exempel accepteras inte en lösning som tar 1 minut för att konkatenera två strängar som är 4 bokstäver långa. Eventuella undantag anges uttryckligen i uppgiften.
- Om inte annat sägs ska funktioner **returnera** värden, inte skriva ut dem.

Avdrag för felaktig och ofullständig kod ges **även om lösningen är uppenbar** för granskaren, och även om det syns att lösningen "kan ha varit på rätt väg". Det ingår i uppgiften att se till att koden uppfyller specifikationen, och att *själv* upptäcka och åtgärda problem under tentans gång. Den inlämnade koden är ditt slutgiltiga svar, inte ett mellansteg som granskaren ska arbeta vidare med.

Lösningar som misslyckas i alltför många fall räknas normalt inte som lösningar och får 0 poäng. Det går oftast att få delpoäng för lösningar som *misslyckas* med vissa specifika fall, men inte för lösningar som *bara lyckas* med vissa specifika fall.

Tillåtet / icke-krav

Vi får ofta frågor om vad som är tillåtet i en lösning. **Om inget annat anges** i en uppgift, är följande uttryckligen **tillåtet**:

- Att lösa uppgiften **rekursivt eller iterativt, eller med en kombination** av dessa lösningsmodeller (t.ex. hybrider med rekursiva anrop och defaultargument). Med andra ord: Om inget annat sägs kräver vi inte någon specifik form av rekursion, och du kan använda en lösningsform du känner dig bekväm med.
- Att importera och använda **alla vanliga "inbyggda" funktioner** från Pythons standardbibliotek (upp till och med Python 3.11), t.ex. matematiska funktioner från `math`, högre ordningens funktioner som `map()`, och så vidare.
- Att använda **listbyggare** (list comprehensions), generatoruttryck (generator expressions), slicing (delsekvenser såsom `seq[2:5]`) och andra funktionaliteter i språket.
- Att skapa **hjälpfunktioner**, nästlade eller icke nästlade. Lösningen i sin helhet måste så klart fortfarande följa den tänkta lösningsmodellen, om en sådan är angiven.
- Att **addera defaultargument till funktioner**, så länge som funktionerna fortfarande går att anropa på sådant sätt som visas i uppgiften. Lösningen i sin helhet måste så klart fortfarande följa den tänkta lösningsmodellen, om en sådan är angiven. Defaultargument kan ibland användas på sätt som bryter mot en rekursiv lösningsmodell och är alltså problematiska om uppgiften kräver en sådan modell. Se även varningar för defaultargument under rättningskriterier.
- Att **anta att indata följer specifikationen i uppgiften**, utan några egna felkontroller. Står det t.ex. att funktionen ska ta en sekvens, behöver man inte själv kontrollera att man faktiskt får en sekvens som parameter (om inte uppgiften särskilt anger detta). Står det att funktionen ska ta en lista av heltal, får man anta att det är en lista och att den bara innehåller heltal.

  Funktioner måste alltså ge *korrekta svar för korrekta indata* enligt uppgiften, men om inte annat anges får de *krascha eller ge felaktiga svar för felaktiga indata* (såsom när en funktion som bara ska hantera heltal ges en sträng som parameter).
- Att **bryta mot "ytliga" kodningsstandarder** i fråga om t.ex. mellanrum, indentering, radbrytningar, radlängd och antal blankrader, så länge som koden fortfarande är *lättläst och lättförståelig*. Samma gäller stil på identifierare (`snake_case`, `camelCase` med mera).
- Att **hoppa över "type hinting"** såsom `seq: list[int]` i de funktioner man själv skriver. Vi ger ofta *type hints* i funktionerna som anges i tentan, men det är bara för att hjälpa er se vilka indata som förväntas.

Detta gäller inte om uppgiften gör specifika undantag!