INDU: Individuell uppgift i programmering
Generella instruktioner

30 juli 2023

Utvecklat för kurserna DA2004 – Programmeringsteknik för matematiker och DA2005 Programmeringsteknik av Lars Arvestad, Anders Mörtberg, och Kristoffer Sahlin.

Innehåll

Generella krav **1**

Inlämning **1**

Rapporten **1**

Betygssättning **2**
Kriterium 1: Bra kod (6p) 2
Kriterium 2: God felhantering (2p) 3
Kriterium 3: Testning (2p) 3

Redovisning **3**

Generella krav

För alla inlämningar gäller att

- du jobbar individuellt, dvs skriver din egen kod och hittar egna lösningar,
- redovisar hjälp och vägledning du har fått (detta inkluderar inspiration hittad på Internet),
- programmerar i Python 3, och
- att lösningen fungerar utan felmeddelande givet giltig indata.

Som hjälp med programmeringen får man använda alla moduler i Pythons standarddistribution, samt `matplotlib`, `numpy`, och `scipy`. Om det finns ytterligare moduler som skulle kunna vara praktiska så bör man fråga kursledare först.

Inlämning

Du ska lämna in din kod tillsammans med en kort rapport på kurswebben. Rapporten måste vara en PDF (filformat `.pdf`) för att garantera att alla kan läsa den. Koden måste vara enkel att testa och var noga med att skicka med eventuella data- och testfiler.

Rapporten

Rapporten ska inte vara lång, bara funktionell. **Maxlängd är 3 sidor**. Man får ha vilken font, fontstorlek, marginalstorlek, etc., som man vill så länge rapporten är läsbar.

Rapporten **måste** innehålla:

1. Vilken uppgift du implementerat.
2. Hur programmet startas och används.
3. Vilka bibliotek/moduler som används och hur dessa hämtas hem och installeras om de inte är en del av Pythons standarddistribution.
4. Beskrivning av hur du strukturerat ditt program (vilka filer innehåller vad, vilka klasser finns och vad är deras syfte, etc.).

Rapporten **kan** även innehålla reflektioner kring:

- koddesign,
- vilka algoritmer som används och varför,
- vilka datastrukturer som används och varför,
- tids- och minneseffektivitet hos koden.

Obs: det främsta syftet med rapporten är att underlätta för de som ska använda och testa er kod. Om man inte har med en rapport som innehåller de fyra obligatoriska punkterna ovan (1.–4.) är projektet underkänt.

Betygssättning

För att kunna få godkänt krävs ett program som löser den formulerade uppgiften och uppfyller generella (se ovan) och projektspecifika krav. Man måste även ha med en rapport enligt ovan.

För högre betyg måste man uppfylla betygshöjande kriterier. Vissa projekt kan bara ge betyg E-C och andra kan ge E-A, beroende på svårighetsgrad, vilket framgår av uppgiftlydelsen.

De tre kriterier som höjer betyget är:

1. Bra kod (tre delkriterier, 2p styck, så 6p totalt)
2. God felhantering (2p)
3. Testning (2p)

Dessa kriterier kommer poängsättas med 0-2 poäng enligt:

0. Uppfyller inte kriteriet.
1. Uppfyller kriteriet.
2. Uppfyller kriteriet väl.

Vilket betyg man får för projektet, baserat på vilket projekt man gjort samt hur många poäng (0-10) man fått, sammanfattas av den här matrisen:

Poäng:	0-2	3-4	5-6	7-8	9–
Enklare projekt	E	D	C	C	C
Svårare projekt, uppg. 1	E	D	C	C	C
Svårare projekt, uppg. 1 & 2	E	D	C	B	A

Nedan följer förtydligande vad kriterierna innebär.

Kriterium 1: Bra kod (6p)

För att uppfylla det här kriteriet måste koden följa instruktionerna i ”Tumregler för programmering” och delen om ”Bra kod” i kompendiet. Man bör även ha lämplig uppdelning i klasser, dvs programmet bör vara skrivet i objektorienterad stil.

Poängen för det här kriteriet delas upp i 3 delkriterier:

A. Dokumentation (2p)

   - Lämplig mängd informativa kommentarer som förklarar viktiga steg och antagande i koden.

- Alla funktioner, klasser och metoder ska ha en dokumentationssträng som förklarar vad deras syfte och funktion är utöver vad som redan framgår av den valda identifieraren.

B. Läsbarhet (2p)

- Väl valda och informativa namn på identifierare.
- Lämplig radlängd (helst under 80 tecken, definitivt under 100).
- Lämplig längd på funktioner och metoder (max en halv laptopskärm).

C. Struktur och objektorientering (2p)

- Meningsfull och bra uppdelning i klasser. För att uppfylla det här kriteriet väl **måste** koden vara uppbyggd på ett objektorienterat sätt. Majoriteten av koden ska alltså ligga i lämpliga klasser och endast ett fåtal funktioner på toppnivå (exemplvis en huvudfunktion, `main`).
- Bra uppdelning i metoder/funktioner som gör att kodduplikation undviks.
- Tydlig filstruktur och organisation av koden.
- Tydlig separation av metoder/funktioner som utför beräkningar och de som utför användarinteraktion. Det betyder att man har särskilda funktioner/metoder för beräkningar som ej innehåller satser som `print`, `open`, `input`, etc.
- Funktioner ska inte vara beroende av globala variabler, men man får använda sig av globala konstanter (dvs variabler som inte ändrar värdet någonstans i programmet) om det är lämpligt. Om man använder några globala konstanter måste man motivera dessa i kommentarer.

Kriterium 2: God felhantering (2p)

Programmet får inte krascha för vissa körningar, vare sig för slumpeffekter som uppstår i simuleringar, beroende på felaktiga parametrar eller olämpliga indata från användaren, felakigt innehåll i filer, etc. För att undvika detta måste ni använda lämplig felhantering genom `try-except` block. Ni bör även lyfta lämpliga särfall med hjälp av `raise` när det behövs.

För att uppfylla detta kriterium väl måste felhanteringen skötas på ett bra sätt så som beskrivs i kompendiet. Till exempel bör man inte lyfta ett särfall för att sedan direkt fånga det eller använda `try-except` när det vore mer passande med en `if`-sats.

Kriterium 3: Testning (2p)

Varför ska granskaren lita på att programmet fungerar? Att förklara i sin rapport att "det verkar fungera" duger inte, utan du ska motivera varför granskaren ska lita på att ditt program fungerar. För att uppfylla detta kriterium väl **måste** man använda enhetstester med hjälp av `unittest`.

För att kunna testa dina metoder/funktioner på ett meningsfullt måste de vara skrivna på ett sätt som gör det möjligt att faktiskt testa dem. Om du har svårt att skriva bra tester är det ett tecken på att du bör tänka om designen av ditt program. En bra tumregel är att genom att skriva korta funktioner som *returnerar* ett resultat kan man mycket enklare skriva tester som verifierar att allt fungerar som man tänkt sig.

Redovisning

I vanliga fall krävs ingen redovisning, men vid oklarheter kan vi kräva en muntlig redovisning. Under redovisning måste man kunna svara på frågor om lösningen muntligt.