INDU—Individuell Uppgift i Programmering

18 februari 2022

Utvecklat av Lars Arvestad för kursen DA2004 - Programmeringsteknik för matematiker. Modifierad av Kristoffer Sahlin och Anders Mörtberg.

Innehåll

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,
- programmerar i Python 3, och
- att lösningen fungerar utan felmeddelande givet giltig indata.

Efter inlämning kommer en granskningsfas då du ska granska kamraters lösningar, precis som vi gjort med laborationerna. Du kan inte bli godkänd på INDU om du inte har granskat tre andra inlämningar.

Som hjälp med programmeringen får man använda alla moduler i Pythons standarddistribution, samt `matplotlib`, `numpy`, `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 PeerGrade.

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. Hur du strukturerat ditt program (vilka filer innehåller vad, etc.).

Rapporten **kan** även innehålla reflektioner kring:

- koddesign,
- vilka algoritmer som används och varför,
- vilka datastrukturer som används och varför,
- tid 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 få betyg E krävs ett program som löser den formulerade uppgiften och uppfyller generella (se ovan) och projektspecifika kraven. 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.
2. God felhantering.
3. Testning.

Dessa kriterier kommer poängsättas med 0-2 poäng enligt:

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

Betyg och projektmål är sedan:

A. Mycket bra och tydlig kod. Mycket bra felhantering och testning. (6 poäng totalt)
B. Mycket bra och tydlig kod. Det finns felhantering och testning. (4-5 poäng totalt, varav 2 på kriterium 1)
C. Koden är av bra kvalitet. Det finns felhantering och/eller testning. (3 poäng totalt, minst 1 på kriterium 1)
D. Koden är av bra kvalitet, eventuellt med felhantering eller testning. (2 poäng totalt, minst 1 på kriterium 1)
E. Koden fungerar och löser uppgiften. (0-1 poäng totalt eller 0 poäng på kriterium 1)

Nedan finns en matris som sammanfattar vilket betyg man får på projektet baserat på vilket projekt man gjort samt hur många poäng (0-6) som man fått på de betygshöjande kriterierna:

Poäng:	0	1	2	3	4	5	6
Enklare projekt	E	E	D*	C*	C*	C*	C*
Svårare projekt, uppg. 1	E	E	D*	C*	C*	C*	C*
Svårare projekt, uppg. 1 & 2	E	E	D*	C*	B**	B**	A

C*/D*: För att få betyg C/D krävs minst 1 poäng på kriterium 1.
B**: För att få betyg B krävs 2 poäng på kriterium 1.

Nedan följer förtydligande vad kriterierna innebär.

Kriterium 1: Bra kod

För att uppfylla det här kriteriet måste koden följa instruktionerna i ”Tumregler för programmering” och delen om ”Bra kod” i kompendiet. Detta innebär att man bland annat ska ha:

- Väl valda och informativa namn på identifierare.
- 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.
- Lämplig radlängd (helst under 80 tecken, definitivt under 100).
- Lämplig längd på funktioner och metoder.
- Tydlig filstruktur och organisation av koden.
- Bra uppdelning i funktioner, metoder och klasser så att kodduplikation undviks.
- Tydlig separation av funktioner/metoder 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 bör inte vara beroende av globala variabler om det ej är nödvändigt. Om ni använder några globala variabler måste ni motivera dessa i kommentarer.

Kriterium 2: God felhantering

Programmet får inte krascha för vissa körningar, vare sig för slumpeffekter som uppstår i simuleringar eller beroende på felaktiga parametrar eller olämpliga indata från användaren. 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

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 (m.h.a. `unittest`).

För att kunna skriva bra tester måste funktioner och metoder vara skrivna på ett sätt som gör det möjligt att testa dem på ett meningsfullt sätt. 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.