CHALMERS
EXAMINATION / TENTAMEN

Course code/kurskod	Course name/kursnamn			
DAT555	Programmeringsteknik			
Anonymous code Anonym kod		Examination date Tentamensdatum	Number of pages Antal blad	Grade Betyg
DAT555-0009-ZFY		5/1 - 23	11	5

* I confirm that I've no mobile or other similar electronic equipment available during the examination.
Jag intygar att jag inte har mobiltelefon eller annan liknande elektronisk utrustning tillgänglig under eximinationen.

Solved task Behandlade uppgifter No/nr		Points per task Poäng på uppgiften	Observe: Areas with bold contour are to completed by the teacher. Anmärkning: Rutor inom bred kontur ifylles av lärare.
1	X	3	
2	X	4	
3	X	5	
4	X	3	
5	X	3	
6	X	6	
7	X	4	Σ28
8	X	6	
9	X	5	
10	X	5	
11	X	5	
12	—	—	
13			
14			
15			
16			
17			
Bonus poäng			
Total examination points Summa poäng		49	

4

A1) Ett if statements används för att testa ett påstående och om påståendet gäller utföra en viss åtgärd

If är ett statement som innebär att när det exekveras så åstadkommer en effekt tex:

```
x = 5
for i in range(x+1)
    if  x < 10:
        x += 1
    elif x < 15
        x += 2
    else:
        x += 3
```

I ett if statment kan som visat ovan vara if, som följs med elif eller avslutningsvis else som innebär att om inget ~~av~~ av påståndena ovan stämmer så exekveras denna, "annars". ~~Man kan använda if~~
~~Det är vanligast att använda if först~~
Man börjar med if och efter det om man ska testa ett/flera specifika påståenden efter är det vanligt att göra/använda elif.

Påståenden som if testar kan vara som ovan om något är mindre än något och if är ingen loop eller iteration men man kan testa flera påståenden i en loop tex för om man vill loopa igenom tex en lista och testa påståenden. Man använder alltså inte while då som inte loopar igenom en lista med påståenden. ~~för~~ eller om man vet hur många gånger man ska loopa.

Vi vill alltså använda för att testa ett påstående och om det gäller ~~eller~~ utföra en effekt

3

4

A2)

{x = 5}

5+1 = x+1

try:
  for i in range(6)
    x += 1 + (i // 5 - i))

här hoppar vi när i = 0, ~~sedan i = 1, i = 2, i = 3, i = 4, i = 5~~

x += 1 + (0 // 5) → {x = 6}

när i = 1:

x += 1 + 1 // 4 → {x = 7}

i = 2: x += 1 + 2 // 3 → {x = 8}

i = 3: x += 1 + 3 // 2 → {x = 10}

i = 4 → x += 1 + 4 // 1 → {x = 15}

i = 5 → x += 1 + (5 // 0) Här är ett division med 0 som

innebär att ett exception inträffar som sedan leder till att except-blocket först exekveras, där det första except som matchar exekveras, här är det Except ZeroDivisionError: som följer x += 1

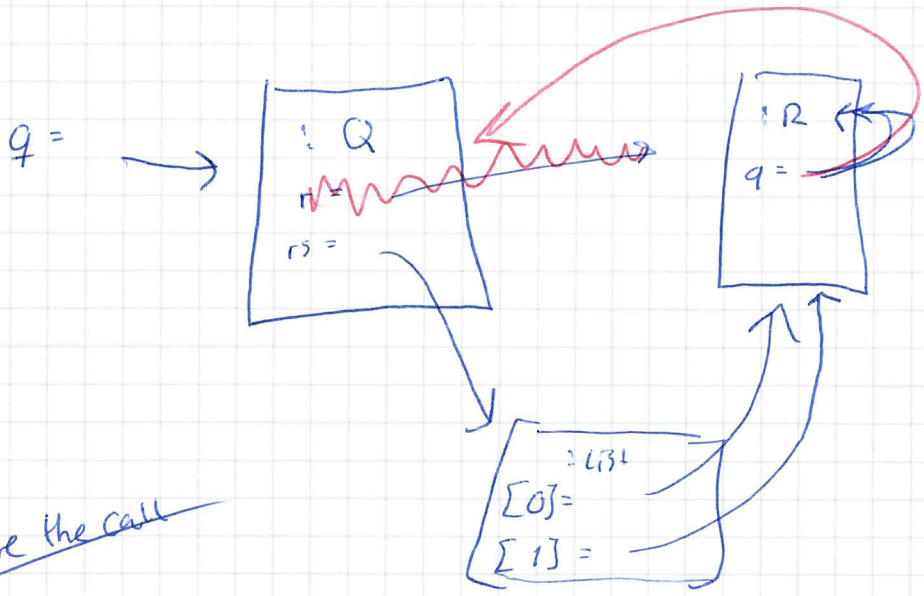
vilket ger oss x = 21, vilket sedan är det som printas

4

CHALMERS | Anonymous code | Points for question (to be filled in by teacher) | Consecutive page no. Löpande sid nr 3

Anonym kod: DAT555-0009-ZFY | Poäng på uppgiften (ifylles av lärare) 5 | Question no. Uppgift nr 3

A3)



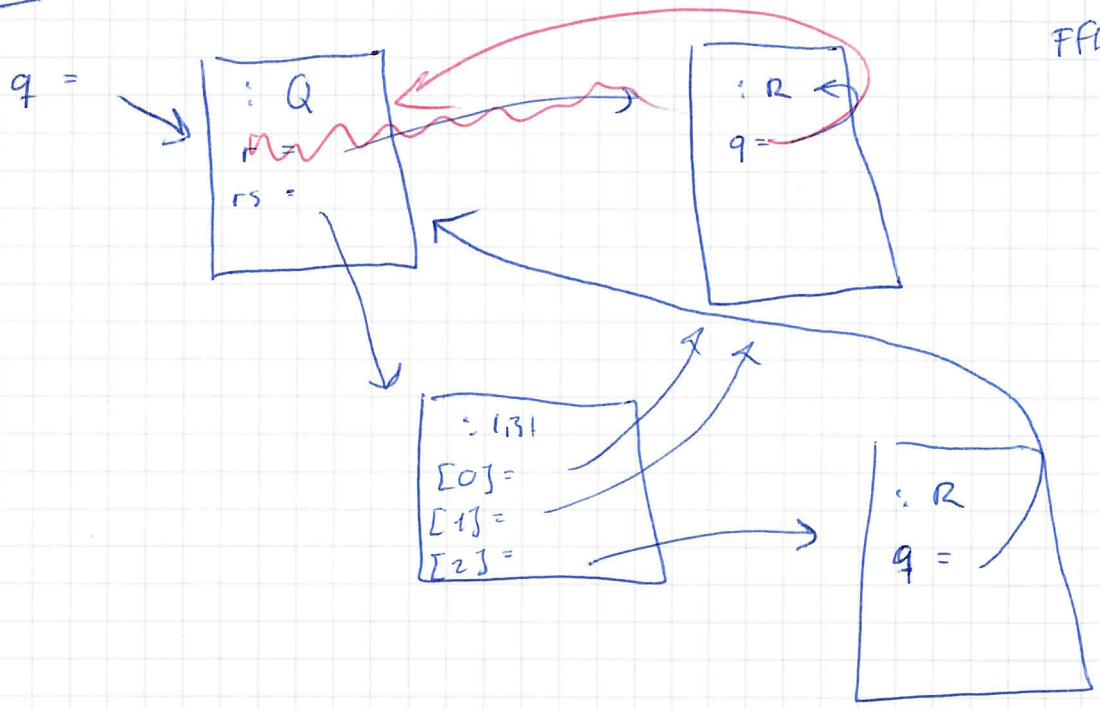
Before the call

For

for i in range(2) i=0, i=1

The call

Fflr



5

4

A4)

~~def products(in_list):~~
~~out_list = []~~
~~tmp_list = []~~
~~if in_list > 1:~~
~~tmp_list.append(in_list[0])~~
~~for elem in in_list[1:]~~
~~tmp_list.append(elem)~~
~~for elem in tmp_list:~~
~~add_it = elem * tmp_list[-1]~~
~~out_list.append(add_it)~~
~~break~~
~~return out_list~~

```
def products(in_list)
    out_list = []
    tmp_list = []
    if len(in_list) > 1:
        tmp_list.append(in_list[0])
        for elem in in_list[1:]
            tmp_list.append(elem)
            for elem in tmp_list:
                add_it = elem * tmp_list[-1]
                out_list.append(add_it)
                break
    return out_list
```

Shadowing - vilket här ej blir buggg

Vad är poängen med en loop som alltid breakar?

→ gå till första for loopen

3

4

A5) Konstruktor anropas för att skapa en instans av ett objekt. 1
~~Anropar~~ __init__ ← Genom
Det är här man ger objekten de värden som man önskar objektet ska ha
1

Vi vill använda dessa för att ge objekten de specifika egenskaper / värden man önskar och anropas för att skapa en instans av ett objekt
1

3

4

A6) Abstraktion används för att dölja tekniska detaljer och erbjuda ett enklare gränssnitt.

Abstraktion innehar:
1) Mindre information
2) Mindre beroende
3) Inkapsling
4) Gränssnitt
5) Namngivning, (Funktionell nedbrytning, skapa klasser.)

Genom abstraktion så undviker användare att ta del av information på detaljnivå, vars etiska konsekvenser annars lett till att informationen skulle kunna hamna i fel händer.

Genom abstraktion så:
- Kan användare och programmerare komma fram till vad användaren ska kunna göra så kan programmeraren lösa problem och implementera medans användaren använder programmet
- Programmeraren kan ändra i koden utan att påverka användaren
- Det blir tydligt och mer lätt läst för användaren vilket gör att onödig tid på att försöka förstå saker inte läggs.

De etiska konsekvenser blir alltså motsatsen till vad abstraktion hjälper med. Koden blir inte lika lättläst vilket gör att de blir svårt för användaren och framtida programmerare att kunna arbeta med den. Det hade också lett till konsekvenser som att användaren kan ta del av detaljinivå/info något som kan leda till att information som inte skulle komma ut, läcker och hamnar i fel händer. Etiska konsekvenser följer också att programmerare inte kan ändra i koden utan att riskera att berörda användare.

Sammanfattningsvis vill vi programutvecklare arbeta med abstraktion för att dölja tekniska detaljer och erbjuda enklare gränssnitt med namngivning, inkapsling, funktionell nedbrytning, skapa klasser etc.

6

4

A7] Abstrakt metod är en egentligt signatur av metoden, den ~~har~~ benämns @abstract metod, (~~när~~ den deklareras)

~~Anv~~ Används för att signalera att det krävs kompletterig/implementig där subklasser måste ~~skapa~~ implementeras för att skapa instanser, annars blir det fel.

Abstrakta metoder kan skapas i finns i abstrakta klasser pga att man vill inte att objekt skapas av klassen utan skapa objekten av subklasserna.

- gränssnittet, signatur är en metod.

4

4

B1) Klassen för x skapas, inget mer händer

Klassen för Y skapas där x1 skapas och hänvisas till Klassen x där "static in Y" printas. /

Sedan skapas klassen Z där ~~instansobjektet x1 skapas~~

x1 skapas och hänvisas till klassen x där 'statk in Z' /

skrivs.

Sedan körs uppgift B1

där Z, ~~instansobjektet~~ skapas till Klassen Z där

instansobjektet a2 skapas som hänvisar till ~~K~~klassen X som skickar in "instance in Z" som printas /

Sedan skapas instansobjektet b1 av klassen Y där instansobjektet x2 skapas av klassen x där "instance in Y" printas /

Sedan printas "Z is done Run 1" /

Exakt samma sak händer när det andra Z skapas först

då printas Run 2 i slutet /

6

4

B2) Klass är en specifikation av typ 1
- Definerar hur ett objekt av en specifik typ skapas 1
- Deklarerar att ett objekt också är ett objekt av samtliga supertyper 1
- Deklarerar ett klass-objekt med egen funktionalitet. 1

Kan se det som en produktionsmaskin med instruktioner för en specifik typ av objekt.
Inkluderar instanser och attribut

Så vi definerar en klass, som kan definera ett klassobjekt som skapar när en klass deklarerar och varje klass är ett sådant. Ger specifika egenskaper/värden till objektet och vad de ska kunna göra m.h.a metoder.

Instanser skapas vid anrop av klassens konstruktör och ger objekten de specifika egenskaper man önskar de ska ha. Skapas i den riktning klassen står och har instans och klass-attribut och metoder. Attribut har till objekten och har på heapen och är variabler som refererar till klasser och instanser och ~~Ger objekt~~ håller objektet ~~vadet~~/värdet under programets gång. 1

Kan använda klass när:
- Samlingsnamn för rena funktioner som behöver ett gemensamt hem
- Produktionsmaskin för instanser
- Skapa en specifik server tex Pongguin

5

CHALMERS

Anonymous code / Anonym kod: DA7555-0009-2F4

Points for question (to be filled in by teacher) / Poäng på uppgiften (ifylles av lärare): 5

Consecutive page no. / Löpande sid nr: 10

Question no. / Uppgift nr: 10

B3) Datastrukter är vanliga och viktiga abstraktioner som används för att samla och arbeta med data.
Vilken datastruktur man använder beror på vilket gränssnitt man har/vill ha och syftet bakom användandet.

Finns inbyggda som listor, tupler, dictionary (dict)

5

B4) Arv och komposition handlar om att återanvända kod genom att skapa en hjälpklass med gemensam funktionalitet

komposition innebär att en hjälpklass skapas med gemensam funktionalitet som attribut, metoder etc sedan kan användas genom att de har ett objekt med en referens till denna. Komposition har ett has a relation och så istället för att olika klasser etc skall behöva duplicera den gemensamma funktionaliteten i flera funktioner finns ~~och~~ komposition och en hjälpklass.

Arv används också för att undvika kodduplicering genom att en supertyp skapas / superklass som är och fungerar som en hjälpklass likväl men som sedan har subklasser som ärver av superklassen dess funktionalitet, vilket gör att subklasserna blir beroende av superklassens funktionalitet, har en is a relation.

Fördelar med de båda är att kodduplicering undviks vilket gör det enklare att hänga med och hjälper att endast behöva ändra på ett ställe. Men med arv blir ~~man mer bero~~ subtyperna för ett beroende till superklassen vilket gör att det subklasserna
kan begränsa subtyperna om det är beteenden / funktioner som subklassen inte vill ha men som den ärver ~~tex~~
komposition är på detta sätt inte skapar lika starkt beroende.

Med arv gäller en ägaren där det ägda vilket inte är fallet med komposition.

Tex om vi har i Arv en supertyp Animals() och den har funktioner som sing(), fly() etc så ärver subtypen Pingvin och tex sköldpadda av denna, men sköldpaddan vill inte ha dessa funktioner då det inte stämmer.

Så är det bättre med komposition där en hjälpklass kan skapas för allt som är likt för alla djur

5