CHALMERS
EXAMINATION / TENTAMEN

Course code/kurskod	Course name/kursnamn			
DAT555	Programmeringsteknik i Python			
Anonymous code Anonym kod		Examination date Tentamensdatum	Number of pages Antal blad	Grade Betyg
DAT555-0022-CAE		16/08/23	11	4

* 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	2	
2	X	4	
3	X	6	
4	X	5	
5	X	3	
6	X	5	
7	X	3	Σ28
8	X	1	
9	X	3	
10	X	4	
11	X	6	
12	—	—	
13			
14			
15			
16			
17			
Bonus poäng			
Total examination points Summa poäng på tentamen		42	

5

CHALMERS	Anonymous code Anonym kod DAT555-0022-CAE	Points for question (to be filled in by teacher) Poäng på uppgiften (ifylles av lärare) 2	Consecutive page no. 1 Löpande sid nr Question no. A1 Uppgift nr

A1. Ett return statement returnerar ett värde eller ojekt.

Det används till exempel när en variabel genom en funktion ändrat värde eller pekar på ett nytt objekt

Ex. def add_five(x)
    return x+5

I detta exemplet kommer argumentet som skickas in adderat med 5 vara värdet som returneras.

return används alltså generellt för att få resultatet av en funktion

2

Answer only one question on this page. Do not write on the back of this paper
Behandla endast en uppgift på detta blad. Skriv ej på baksidan

5

CHALMERS	Anonymous code Anonym kod DAT 555-0022-CAE	Points for question (to be filled in by teacher) Poäng på uppgiften (ifylles av lärare) 4	Consecutive page no. Löpande sid nr 2 Question no. Uppgift nr A2

A2.

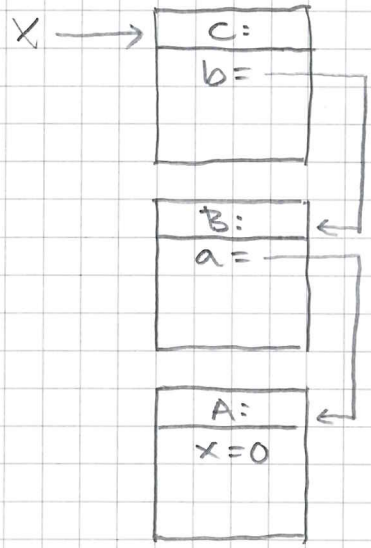
1. x sätts till 0
2. x < 12 då 0 < 12 och x går in i while-blocket
3. 1 adderas till x. x = 1
4. 1 är inte jämt delbart med 3 och x hamnar i else som säger continue
5. Continue skickar tillbaka x till närmast omslutande loop, alltså while-blocket
6. x < 12 så x är kvar i while-blocket
7. 1 adderas till x. x = 2
8. 2 är inte jämt delbart med 3 och skickas till else som säger continue
9. x börjar om i while-loopen då 2 < 12
10. 1 adderas till x. x = 3
13. 3 är jämt delbart med 3 (har resten 0) och x skickas in i if-blocket
14. x multipliceras med 2. x = 6
16. 4 adderas till x. x = 10
17. 10 < 12 så x fortsätter i while-blocket
18. 1 adderas till x. x = 11
19. x = 11 är inte jämt delbart med 3 och x fortsätter till else-blocket som säger continue
20. 11 < 12 så x fortsätter i while-blocket
21. 1 adderas till x. x = 12
22. 12 är jämt delbart med 3 och går in i if-satsen
23. x multipliceras med 2. x = 24
24. 4 adderas till x. x = 28.
25. x > 12 så x går ur while-blocket
26. x printas till 28

4

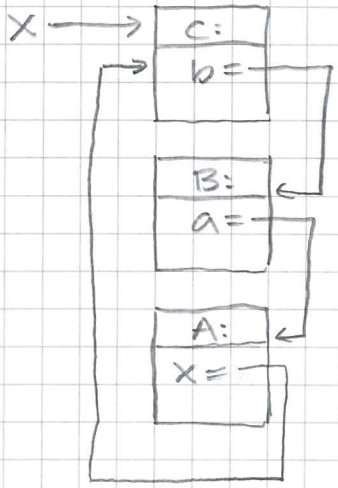
5

CHALMERS	Anonymous code Anonym kod DAT555-0022-CAE	Points for question (to be filled in by teacher) Poäng på uppgiften (ifylles av lärare) 6	Consecutive page no. Löpande sid nr 3 Question no. Uppgift nr A3

A3



do_it:



6

5

CHALMERS	Anonymous code	Points for question (to be filled in by teacher)	Consecutive page no. Löpande sid nr 4
	Anonym kod DAT555-0022-CAE	Poäng på uppgiften (ifylles av lärare) 5	Question no. Uppgift nr A4

A4

```
def validate_intervals(in_list):
    if len(in_list) > 0:
        for i in in_list:
            if in_list[i][0] < in_list[i][1]:
                return True
            else:
                return False

        for k in in_list[0:-2]
            if in_list[k][1] < in_list[k+1][0] and
            in_list[k][0] < in_list[k+1][0]:
                return True
            else:
                return False
```

early exit! continue

continue

Om båda for-block returnerar True så returnerar funktionen True annars returnerar den False

5

CHALMERS	Anonymous code Anonym kod DAT555-0022-CAE	Points for question (to be filled in by teacher) Poäng på uppgiften (ifylles av lärare) 3	Consecutive page no. Löpande sid nr 5 Question no. Uppgift nr A5

A5.

Attribut är egenskaper hos klasser och används för att ge klassen dessa egenskapet.

Ex.
```
class Person():
    species = "homo sapiens"
    def __init__(self.name):
        self.name = name
```

Denna klassen har attributet "name" och gör att instanser av klassen kan ha egenskapen namn.

Ex.
```
person1 = Person("Bob")
Person2 = Person("Gunn")
Print(Person1.name)  ger nu output: "Bob"
```

Detta var exempel på instansattribut men det går också att ha attribut som tillhör alla instanser av klassen och där alla är samma. Både person1 och person2 har attributet species som säger att de är homo sapiens, samtidigt som de båda har namn men dessa är olika.

3

5

CHALMERS	Anonymous code	Points for question (to be filled in by teacher)	Consecutive page no. Löpande sid nr 6
	Anonym kod DAT555-0022-CAE	Poäng på uppgiften (ifylles av lärare) 5	Question no. Uppgift nr A6

A6. Exceptions används för att "fånga" fel i programmet. Detta görs för att programmet inte ska krascha om kod eller input är felaktig.

Med exceptions kan man fånga ett flertal olika errors, tillexempel ZeroDivisionError och TypeError

Man fångar dessa fel med try-except block som exekverar koden try-blocket och om något är fel, tillexempel att man försökt dela med noll, inputat str istället för int för att addera värden eller direkt försökt använda en abstrakt klass vars metod ej är implementerad, så fångas dessa i except-blocket.

Ex.
```
try:
    x = input("input value:")
    b = 2/x
except ZeroDivisionError
    print("can't divide by zero")
```

Fler?

Här fångas felet om inputen är 0 och gör att programmet inte kraschar.

Att inte använda exceptions enligt rekommenderade principer gör att programmet har stor risk för buggar. Det gör det också svårare för ägaren av programmet eller annan programmerare att felsöka programmet. Bra användning av exceptions kan också göra programmet lättare att förstå, underhålla och bygga vidare på.

5

5

CHALMERS	Anonymous code Anonym kod DAT555-0022-CAE	Points for question (to be filled in by teacher) Poäng på uppgiften (ifylles av lärare) 3	Consecutive page no. Löpande sid nr 7 Question no. Uppgift nr A7

A7

En subtyp är en typ som har ärvt något från en supertyp.

Ett bra exempel på en supertyp är en superklass. I superklassen finns attribut som subklasserna sedan kan ärva. Detta gör att koden inte måste skrivas om flera gånger i varje klass som ska ha attributet och minskar därmed kodduplicering. Att använda sub och supertyper kan också göra koden mer modulär, lättare att förstå och felsöka.

Ex.

```
class Car:

    def __init__(self, engine)
        self.engine = engine

class Toyota(car):

    super().__init__(self, engine)
```

Toyota är en subklass av supertypen Car

3

B1.

Steg 1. Först skapas en instans av klassen R som har argumentet 5 och en referens till denna sparas i variabeln r.

r.a har här värdet 8 eftersom argument 5 skickas in. 3 adderas sedan till 5 och r.a = 8 Nej, 9.

2. Nu körs r.f(7) vilket innebär att metoden f(7) som har argumentet 7 körs på instansen R(5).

r.a håller nu värdet 7+4 som är 11. Detta då vi skickar in värdet 7 som argument i metoden f och detta värdet sedan adderas med 4. super call?

3. Print(r.a) # output: 11

4. R.a vill printa a av R då R inte fått några argument (R()).
a sätts till 1 i P, sedan adderas 1 till a i Q (a=2). I R adderas a med 3 och a blir slutligen 5.

Print (R.a) ger output 5 ?

1

CHALMERS	Anonymous code	Points for question (to be filled in by teacher)	Consecutive page no. Löpande sid nr 9
	Anonym kod DAT555-0022-CAE	Poäng på uppgiften (ifylles av lärare) 3	Question no. Uppgift nr B2

B2.

Alias är när två variabler pekar på samma ~~värde eller~~ objekt. Ex x=5 och z=x Nej eller numbers = [1,2,3] och numbs = numbers. Detta gör att en ändring i ena variabeln också innebär en ändring i den andra vilket kan skapa förvirring när vi inte tänker på att båda uppdateras.

För att skydda sig mot detta kan man konsekvent enbart använda sig av en av variablerna. Man kan också se till att dessa ej används i funktioner där de ändras.

Alias kan till exempel användas i listor när man vill ha samma objekt eller värde i listan flera gånger:

```
L1 = [1,2]
L2 = [0]
my_list = [L1, L2, L1]  # här är L1 alias till varandra
```

Om man sedan ändrar något i en av L1 kommer förändringen att ske i båda.
Ex. my_list[0].append(3) kommer båda L1 att uppdateras. Detta måste man vara medveten om och om man inte vill att båda ska ändras är det bättre att ha två likadana listor istället för alias.

3

B3.

Instansmetoder är metoder som tillhör den enskilda instansen och används när instansen kallar på den. Instansmetod tar in första argumentet self och används av enskilda instanser av klassen.

Klassmetoder är metoder som tillhör klassen och måste dekoreras med @classmethod eller @staticmethod. klassmetoder tar in cls som första argument.

Instansmetoderna Initieras när instanser av klassen skapas.

ex.
```
class Person()

    @ classmethod
    def get_species(cls)
        return "Homo Sapien"

    def __init__(self, name)
        self.name = name

    def get_name(self)
        return name

person 1 = Person("Bob")
person 2 = Person("Gunn")
```

person1.get_name() ger output "Bob"
person2.get_name() ger output "Gunn"

Både person1.get_species() och person2.get_species() ger output "Homo Sapien" eftersom klassmetoder kan användas av och är samma för alla instanser av klassen medan instansmetoder tillhör den enskilda instansen och ger olika output för olika instanser av klassen.

Vad bra Hill! 4

B4.

Arv innebär att subtyper ärver funktionalitet (i form av metoder) från en superklass. Detta skapar en hierarki av klasser där det går att ärva i led. Om detta görs i för många led kan koden dock bli svår att förstå, underhålla och utveckla.

Komposition innebär att använda ett objekt som attribut i klassen, ofta klasser.

Ex.
```
class Engine():

    def __init__(self, engine)
        self.engine = engine

    def start_engine(self):
        return "Engine starts"

class Car():

    def engine(self, engine)
        self.engine = Engine()  ← komposition
```

Nu har Car() tillgång till alla metoder i klassen Engine() via attributet engine. Komposition gör koden mer modulär och kan användas när vi vill återanvända kod många gånger utan att skapa förvirrande hierarkier.

Både arv och komposition undviker kodduplicering vilket gör koden lättare att förstå, följa, underhålla och utveckla. Vill man att det ska finnas en hierarki hos klasserna är arv att föredra. Till exempel kan vi ha Ragdoll(cat), Cat(Animal) och Animal(), där Ragdoll är en subklass av Cat som i sin tur är en subklass av Animal. På detta vis kan vi utveckla och specialisera varje klass i led, där Ragdoll har attribut som finns i cat och därför också de från Animal. Komposition ger en möjlighet att modulärt bygga vidare på klasser och kan användas flera gånger med mindre förvirring. Vi kan tex. sätta en Engine() i en Car(), en Truck() eller en Tractor() genom komposition, och bygga upp klasserna på det viset istället. (I Tractor() och Truck() kan vi också sätta Big_wheels() som vi inte sätter i Car().) 6