TDP002 Imperativ programmering

Övningsmaterial

Kursmaterial till kursen TDP002

Höstterminen 2008
http://www.ida.liu.se/~TDP002
Version 1.0

Innehållsförteckning

Introduktion

Detta häfte innehåller en samling övningar för individuellt arbete. Du rekommenderas arbeta med dessa övningar parallellt med labbserien och föreläsningarna. Materialet använder dels övningar från böcker, dels egna övningar (hämtade främst från gamla tentor).

Rekommenderade övningar i läroböcker

Här följer en lista över övningar för kursen Imperativ programmering, indelat efter område.

LP x:y betyder övning ur boken Learning Python Part x övning y.

CS x:y betyder övning ur onlineboken How To Think Like a Computer Scientist http://www.ibiblio.org/obp/thinkCSpy.

EÖ x betyder övning nr x i övningslistan *Extra Övningar* nedan.

Övningar kommer tillkomma under kursens gång i samband med föreläsningarna.

- Köra program (Fö 1-2): LP I:2-7
- Grundbegrepp (Fö 3): LP II:1-3,7,8,9 III: 1 IV:1,2 CS 3:1-4, 5.1-5.8, 7:1-5 CS 8:1-3
- Objekttyper (Fö 4): LP II: 4-6,10-11, EÖ 4-12
- Kontrollstruktur (Fö 5): LP III:2-4, EÖ 1-3, 13-16
- Funktioner (Fö 6-7): LP IV 3-9

Nytt för Utgåva 3.0 av LP är också att det finns Brain Builder Quiz i slutet av varje kapitel med bifogade svar. Mycket bra start vid inlärning är att gå igenom dessa, och testa av mot svaren. Därefter ta itu med ovanstående övningar.

Extra Övningar

Övning 1: Prova med olika exempel så att du förstår hur de grundläggande Booleska operatorerna and, or och not fungerar i detalj. Gör t ex följande interaktiva test:.http://courses.cs.vt.edu/%7Ecs1104/BuildingBlocks/Boolean.test.htm. Prova att skriva in olika uttryck i Python-prompten och se att du förstår varför deras värden blir som de blir.

Övning 2: Gör sanningstabeller (på papper) fast nu för följande tre uttryck:

a or (b and c)

not (a or b)

a or ((not b) and c)

Övning 3: använd omskrivningslagarna De Morgan och distributiva lagen (se föreläsningsbilder) och omvandla uttrycken i övning 2. Gör sanningstabeller för de nya uttrycken och verifiera att de har samma tabellvärden som uttrycken i övning 2.

Övning 4: Gör en modul identity.py vars uppgift är att tilldela ett godtyckligt Python-objekt en identitet i form av ett heltal. Modulen ska stödja att man kan slå upp objekten baserat på denna

identitet senare. Det ska gå att ta bort ett objekt utan att andra objekts identitet förändras (dvs lagring i en rak lista med index som identitet fungerar inte). I grundversion är det inget krav att alla konsekutiva (direkt efterföljande) heltal utnyttjas, speciellt vid borttagning kan det bli "hål". Hur gör man för att undvika hål? Gör gärna en utvidning där alla identiteter ligger konsekutivt som heltal efter varandra.

Övning 5: Skriv ut konstanten Pi med olika noggrannhet i en tabell. För varje rad x skriv ut Pi med x antal decimaler. Använd math-modulen för att göra detta.

Övning 6: På föreläsningen visades ett plottningsprogram för sinus-kurvan. Gör en utvidgad version av denna modul där användaren kan välja om man vill plotta sinus, tangens eller cosinus.

Övning 7: Prova att skriva ut alla unicode-tecken mellan 0 och 500. Gör en tabell som skriver ut de tecken som motsvarar "latin-1" (ISO-8859-1) och deras motsvarande unicode-siffror som hexkod. Sök på Internet efter tabeller över dels Unicode, dels latin-1.

Övning 8: Komplettera tabellen i övning 7 med en kolumn för UTF-8-kodningen för tabellens tecken.

Övning 9: Använd Pythons inbäddade strängvariabler (%-variabler) för att skriva ut följande formatterade strängar:

Skriv ut en sträng med talet 4500.5768797 i tre olika format, med och utan exponent.

Skriv ut en tabell med tio kolumner där du skriver ut alla tal från 0 till 10000. Kolumnerna ska vara fem tecken breda och talen vänstercentrerade

Tilldela följande variabler: length, width, area och owner fyra värden. Skriv ut en mening som anger dessa variablers värden på ett prydligt sätt. Gör detta dels genom att ange variablerna med en tupel efter % används därefter % vars(). Fundera över vad som är för och nackdel med dessa två olika skrivsätt?

Övning 10: Gör ett modul owner.py med som representerar data om personer och dess ägodelar. Ägodelarna ska ha namn, pris och inköpsdatum. Varje person ska ha namn, ålder och en lista av ägodelar. Gör grundläggande funktioner för att tilldela en person en ny ägodel och ta bort den igen. Det ska även finnas en funktion som gör att en person kan sälja en ägodel till en annan. Gör testkod som provar att tilldela personer ägodelar och ta bort dem igen, samt säljer ägodelar.

Övning 11: Många verktyg i ett system som Win32 returnerar en lista av dictionary som resultat. T ex:

```
[{'name':'comp1','comment':'room 100','type':WKSTATION,
 {'name':'comp2','comment':'room 101','type':SERVER}]
```

Skriv en funktion som konverterar en sådan lista till en dictionary där varje nyckel har en lista av värden. För exemplet ovan: {'comp1':['room100',WKSTATION],'comp2':['room 101',SERVER]}. Gör sedan en utskriftsrutin som skriver ut en sådan dictionary så att utskrifterna skrivs ut baserat på alfabetisk ordning för nycklarna i tabellen.

Övning 12: Gör en översättare som översätter siffror mellan 0-99 till motsvarande svenska ord. T ex översätts 17 till "sjutton", 33 till "trettiotre" etc. Gör översättaren generell så att den relativt enkelt kan generaliseras till godtyckligt stora tal. (Denna översättning motsvarar det vi kallat generering på föreläsningen - dvs detta är intressant när vi ska presentera resultat för användaren)

Övning 13: Använd modulerna date och datetime i en funktion som räknar fram en lista av tidpunkter räknat från nutid. Listan bestäms av fem indata-parametrar som anger ett tidsintervall må, dag, tim, min samt ett tal som anger hur många värden som ska returneras. Värdena returneras som strängar på lämpligt datumformat.

Övning 14: Skriv följande funktioner som arbetar på listor av listor av objekt:

subcounts(x): gör en funktion som returnerar av alla partiella uppräkningar av elementen i en given lista. T ex: [1,2,3] ger resultatet [[1], [1,2], [1,2,3]]

(svår) ordered_sublists(x): gör en funktion som returnerar listan av alla ordnade dellistor för en given lista av heltal. T ex, x=[2,1,3] och ordered_sublists(x) ska returnera [[1],[2],[3],[1,2],[1,3],[2,3],[1,2,3]] (Vilken ordning sublistorna kommer i spelar ingen roll)

(svår) merge_sublists(x,y): slå ihop sublistor i listorna x och y om de innehåller två identiska element. T ex x=[[1,3],[2,4]] och y=[[3,5],[2,8]] och merge_sublists ska returnera listan [[1,3,5],[2,4,8]] (ordningen i sublistorna spelar ingen roll)

Övning 15: Antag att vi vill använda dictionaries för information om musik-låtar, t ex artist, låtnamn, album, kategori och årtal. Gör följande funktioner som hittar alla låtar:

av en viss artist/visst album/viss kategori

från en viss tidsperiod (mellan två årtal)

rangordnat enligt en specificerad rangordningslista och slumpfunktion. Ranglista för artist: Beatles 0.6, The Who 0.2, Jimi H 0.1, etc.

Övning 16: (a) Gör ett program som skriver ut en sanningstabell för "och" (eng. and) i konsolen. (b) Generalisera (a) så att ditt program skriver ut valfri sanningstabell för de Booleska operatorerna och, eller, icke, implikation, icke-och. Implikation kan skrivas som -> eller imp, dess betydelse är a -> b är detsamma som "om a är sant måste b vara sant". Det kan också sägas som "icke a eller b". Med icke-och (eng nand) menas not (a and b) eller not a or not b. Detta är en vanlig operation i artimetiska kretsar för att bara behöva icke och eller. Separera ut källkoden som skapar sanningstabellen från den som skriver ut den.

Övning 17: Gör en ADT person som innehåller id, namn, ålder. Gör följande funktioner för denna ADT:

find_person_with_id(L, id): hitta person med givet personid.

select_all_persons_older_than(L, age): returnerar alla personer i given lista som är äldre eller lika med åldern given av age.

Övning 18: utvidga person i övn 17 så att varje person även innehåller en lista av referenser till andra personer i sin familj (föräldrar och syskon) och sina vänner. Gör följande funktioner:

find_relatives(plist, p): hitta alla släktingar (familjens familj...) till p
find_family_friends(plist, p): hitta alla vänners vänner som också är släkt med p.

Övning 19: Skriv ett program som räknar antal kodrader i en källkodsfil. Rader som bara är kommentarer ska ignoreras, likaså blankrader. Kan vidareutvecklas med:

- Rader inuti strängar, som börjar med # (och alltså ser ut som kommentarsrader), ska inte

ignoreras.

- Doc-strängar kan ses som kommentarer, dessa ska därmed också ignoreras. Eventuellt kan en parameter till rad-räknaren ange huruvida rader i doc-strängar ska räknas eller inte.
- Räkna satser istället för rader. Rader som innehåller flera satser, med `;` mellan dem, ska alltså räknas flera gånger. Även if-satser skrivna på en rad, och funktionsdefinitioner som skrivits på en rad, ska räknas flera gånger.

Övning 20: Skriv ett program för att lagra information om filmer. Titel, genre och betyg (på någon graderad skala), eventuellt även annan information. Programmet ska kunna:

- lägga till ett omdömme för en film
- lägga till nya filmer
- söka efter filmer med ett minsta medelbetyg.

Exempel hur det kan se ut på skärmen:

```
$ ./rate.py "Die Hard 3" 1 "Alla actionscener från ettan pluss
alla scener från tvåan, repetitivt och tråkigt"
$ ./rate.py "Tank Girl" 4 "Nästan lika bra som serierna"
$ ./rate.py --search --grade 4
Tank Girl - Grade 4.6
Hitchhikers guide to the galaxy - Grade 5.0
$ ./rate.py --info "Hitchhikers guide to the galaxy"
Grade 5: En suverän nytappning.
Grade 5: Den bästa filmatisering av en radioshow jag någonsin
sett.
```

Övningar tagna från tentor 2007-2008

Uppgift 1-071018: Skriv en funktion `format_by(key, personlist)` som tar namnet på en nyckel `key` att formatera efter samt en lista `personlist` med data om personer som argument. Parametern `personlist` är av typen lista av dictionary. Varje dictionary förväntas hålla följande nycklar för en person: `name`, `address`, `birthdate`. Värden för namn och adress är strängar, och för födelsedatum siffror. Samtliga tre nycklar kan förväntas ha värden i varje dictionary. Funktionen `format_by` ska returnera en sorterad lista med värdena på given nyckel, sortera i bokstavsordning för strängvärden. För enkelhetens skulle sorteras även siffror i adressen i bokstavsordning och åäö sorteras baserat på "ASCII-ordning" (där ordningen blir äåö). Bifoga även källkod som visar hur funktionen anropas med en testlista och sortering med olika nyckelvärden. Exempel:

Lista av personer:
Nilsson Ulrika, Huggarvägen 45, 19691201
Person Nils, Adamsvägen 12, 19771013
Olsson Erik, Betongbstigen 3, 19590724

Resultat från `format_by` om vi sorterar på adress:

Adamsvägen 12
Betongstigen 3

Huggarvägen 45

I exemplet har vi inte visat exakt objektstruktur, eftersom det är en del av uppgiften att ta fram själv från uppgiftens text ovan.

Uppgift 2-071018: Gör en högre-ordningens funktion remove_words(remove_test, text) som tar bort alla ord i strängen text för vilka funktionen remove_test returnerar sant. Orden separeras av mellanslag. För enkelhetens skull antar vi att texten saknar andra skiljetecken. Använd remove_words för att göra två funktioner remove_unwanted_words och remove_small_words som ska fungera enligt följande:

```
>>>remove_unwanted_words(['fasen', 'usch', 'fy'], 'usch vad det regnar fy')
'vad det regnar'

>>>remove_small_words(3, 'usch vad det regnar fy')
'usch regnar'
```

Uppgift 3-071018: Gör ett program select_columns.py som läser in en fil med data organiserat i kolumner och väljer ut vissa av dem och skriver ut på en annan fil. Programmet ska ta två argument som anger från och med vilken kolumn och till och med vilken kolumn data som ska väljas ut. Därutöver ska sökväg för in- och utfil kunna anges. I filerna markeras kolumnstart med semikolon (';'). Semikolon kan inte förekomma i filen för övrigt. Programmet ska också göra så att den utfilen får radslut som passar för Windows, dvs \r\n. Om argumenten som anges är på fel format ska felmeddelande ges. Exempel på anrop:

```
% more data.txt
123;hund;4567;ben,godis
-99;katt;55;sand
44.5;ko;333E10;gräs
% python select_columns.py 1 3 data.txt out.txt
% more out.txt
hund;4567
katt;55
ko;333E10
```

(Det syns inte i exemplet ovan att radslutet för out.txt är \r\n vilket det ska vara)

Uppgift 1-080114:

a) Skriv en funktion format_time(seconds) som gör en tid angiven i sekunder till en tupel med fyra element: dagar, timmar, minuter och sekunder. . T ex ska argumentet 73 ge som resultat tupeln (0,0,1,13).

b) Skriv en funktion add_format_time(time1, time2) som adderar två tal representerade i tupelformat enligt a) . T ex argument (0,2,52,44) och (1,0,17,23) bli tupeln (1, 3, 10, 7)

Uppgift 2-080114: Skapa ett program number_file.py som frågar användaren efter namnet på en in-fil och en ut-fil i konsolen, laddar in angiven in-fil och skriver ut filen på angiven ut-fil. Om infilen inte fins eller utfilen inte kan skapas ska felmeddelande skrivas ut i konsolen. Utfilen ska eka ut innehållet på in-filen men här radnummer och antal kolumner i början av varje rad. Kolumnen inom parentes och kolon innan orginalraden börjar. Tomma rader (inkl rader med mellanslag,

tabtecken etc) ska inte ha radnummer. T ex för en infil med detta innehåll:

```
aa
bbb

cccc
```

så ska utfilen se ut så här:

```
1(2): aa
2(3): bbb

3(4): cccc
```

Uppgift 3-080114: Gör en högre-ordningens funktion check_passwords(password_check, dict) som tar en dictionary som par av användarnamn och deras ännu icke-krypterade lösenord. Funktionen ska använda funktionen password_check för att testa om lösenorden uppfyller kraven på ett lösenord. De användarnamn som inte uppfyller testet ska returneras i en lista som resultat. Om alla lyckas returneras en tom lista. Gör sedan två metoder som använder sig av check_passwords och också returnerar felaktiga lösenord på samma sätt som check_passwords:

check_passwords_classic(dict): lösenordet måste vara minst 6 tecken långt och bara innehålla bokstäver och siffror, med minst två siffror och minst två bokstäver.

check_passwords_sloppy(dict): lösenordet ska vara minst 3 tecken långt och måste innehålla minst 3 olika tecken

Vi antar att åääö inte förekommer i användarnamn och lösenord (varken i korrekta eller felaktiga)

Uppgift 4-080114: Skapa ett program make_properties.py som översätter konfigureringsfiler baserat på det "klassiskt Unix-vis" till Pythons ini-format som används på Windows med sektioner. Använd modulen ConfigParser för att generera den nya filen. Ange namnet på infil (enklare unix-format) och utfil (Python format). Det enklare formatet antas vara på en rak lista av med tilldelningar på formen egenskap=värde. Man har använt namnkonvention med punkt för att hantera liknande namn, vilket ska översättas till sektioner. Exempel på infil:

```
global.length=23
global.width=17
global.area.color=red
fallback.color=blue
fallback.time=23
```

Utfilen ska då bli:

```
[global]
length=23
width=17
[global.area]
color = red
[fallback]
color=blue
time=23
```

Uppgift 5-080114: Skapa en modul shopping.py som hanterar en "kundvagn" för Internet-shopping. Du väljer själv den interna representationen för kundvagnen. Följande funktioner ska ingå:

create_shopping_cart(person): skapar en kundvagn för en person med givet namn

add_product(name, quantity price, cart): lägg till produkter till kundvagnen. Kundvagnen ska registrera produktnamn, antal och pris.

lookup_shopping_cart(person, cart_list): hitta och returnera kundvagnen för en person med givet namn i en lista av kundvagnar.

check_out(cart, delivery_cost): Summerar och returnera den totala kostnaden för kundvagnen och lägger till given fraktkostnad.

print_shopping_cart(cart): skriver ut kundvagnens innehåll i konsolen. I utskriften ska ingå: namnet på personen, varornas namn, antal och styckpris samt totalkostnaden för hela kundvagnen

Uppgift 1-080818:

a) Skriv en funktion triple_consecutive(x,y,z) som returnerar True om talen x, y, z är konsekutiva/efterföljande tal. T ex ska anropet triple_consecutive(3,4,5) returnera True men triple_consecutive(5,9,4) och triple_consecutive(5,3,4) returnera False.

b) Skriv en funktion count_triple_consecutives(numb_list) som räknar förekomster av konsekutiva trippler enligt a) . T ex ska count_triple_consecutives([8,1,3,4,5,9]) och count_triple_consecutives([8,1,3,4,5,6]) bägge returnera 1 och count_triple_consecutives([9,1,3,4,5,10,6,7,8]) returnera 2.

Uppgift 2-080818: Gör en högre-ordningens funktion myfilter(filter_fn, alist) som implementerar standardfunktionen filter. Din implementation får inte använda funktionen filter. Gör sedan två metoder som använder sig av myfilter och som fungerar enligt följande:

sum_odd_numbers(alist): returnerar summan av alla udda tal som finns i den givna listan alist.

sum_selected_numbers(selection, alist): returnerar summan av alla tal som finns i både listan selection och i listan alist. T ex ska sum_selected_numbers([3,4,5], [2,5,3,9]) returnera 8.

Uppgift 3-080818: Skapa ett program store_properties.py som läser in konfigurationsdata från användaren i konsolen och sparar ner dem på fil. Inläsningen sker genom att programmet först frågar om egenskapsnamn sedan dess värde tills dess användaren väljer att avsluta dialogen (t ex genom att inte ange någon ny egenskap. Filformatet för konfigurationsdata är en rak lista av med tilldelningar på formen egenskap=värde med en sådan tilldelning per rad i filen. Exempel på utfil:

```
fallback.color=blue
fallback.time=23
global.area.color=red
global.length=23
global.width=17
```

Konfigurationsvärdena ska sorteras alfabetiskt i filen baserat på egenskapsnamnet, se t ex exemplet ovan. Namnet på utfilen ska ges som parameter till programmet. Om inget filnamn ges ska felutskrift göras.

Uppgift 4-080818: Skapa ett program popular_name.py som talar om hur vanligt ett namn är, årtionde för årtionde. Exempel på konsolutskrift:

```
Ange ett namn: Sara
Statistik för namnet Sara:
1900: 58
1910: 74
...
2000: 203
```

Programmet ska använda en fil där data för olika namn sparas. Namnhistoriken ska gå tillbaks åtminstone till år 1900. Du väljer själv lämpligt filformat. Även en exempelfil med data ska skickas med lösningen.