MATEMATISKA INSTITUTIONEN
STOCKHOLMS UNIVERSITET
Avd. Beräkningsmatematik
Kursledare: Anders Mörtberg
Examinator: Lars Arvestad

Tentamen i
DA2005 Programmeringsteknik
7.5 hp
2026-03-14

Note: An English version of this exam starts on page 6!

- Tentan har flervalsfrågor där minst ett svarsalternativ är korrekt. Om man svarar fel eller inte har exakt antal rätta alternativ får man noll poäng på frågan.
- Del A består av flervalsfrågor (totalt 8 poäng).
- Del B består av kodfrågor med varierande poäng (totalt 12 poäng).
- Del C består av kodfrågor inspirerade av labbarna med varierande poäng (totalt 10 poäng).
- Inga import (Pythons standardbibliotek eller externa bibliotek) får användas om de inte nämns eller finns med i uppgiften. Man får använda inbyggda funktioner som len, range och map.
- All kod avser **Python 3**, dvs *inte* t.ex. Python 2.7
- **Hjälpmedel:** Ett A4 med så mycket information du vill. Du får skriva på båda sidorna.
- **Betygsgränser:** E: 15, D: 18, C: 21, B: 24, A: 27, av maximala 30.

Del A: flervalsfrågor (8p)

Var snäll samla svaren på del A på ett svarspapper. Markera inte *dina svar på detta papper!*

1. Betrakta kodsnutten till höger, vad blir värdet på `x` när man har kört koden?

A. 2
B. 4
C. 6
D. 8
E. 10

```
s = "20260314"
x = 0
for c in s:
    d = int(c)
    if d % 2 == 0:
        x += d
    else:
        x -= d
```

2. Vilka av följande tilldelningar gör att `x and (not y and z)` blir `False`?

A. `x = True`, `y = False`, och `z = True`
B. `x = False`, `y = False`, och `z = True`
C. `x = True`, `y = True`, och `z = True`
D. `x = False`, `y = False`, och `z = False`
E. `x = False`, `y = True`, och `z = True`

3. Betrakta kodsnutten till höger, vad är värdet på `out` när den har körts?

A. `Python`
B. `Pyytttthhhhooooonnnnnn`
C. `ytthhhoooonnnnn`
D. `nnnnnnooooohhhhtttyyP`
E. `nnnnnooooohhhtty`

```
s = "Python"
out = ""
i = 0

while i < len(s):
    out += i * s[i]
    i += 1
```

4. Vad används **`break`** för i Python?

A. Bryta en sträng mitt i så att man kan fortsätta den på nästa rad.
B. Avsluta en loop.
C. Kasta särfallet `ProgramBreakError`.
D. Stänga av Spyder.
E. Det finns inget som heter **`break`** i Python.

5. Betrakta kodsnutten till höger, vad blir värdet på `s` när man har kört koden?

A. 3
B. 6
C. 8
D. Inget då ett särfall lyfts p.g.a. att man inte kan ha listor i uppslagstabeller.
E. Inget då ett särfall lyfts då man inte kan beräkna längden på en uppslagstabell.

```
d = { 2 : "hej",
      3 : [1,2,3],
      4 : { 5 : "hej" , 6 : "du" } }

s = 0

for x in d:
    s += len(d[x])
```

6. Vilka av följande påståenden är sanna om Python?

A. Uppslagstabeller kan ha strängar som nycklar.
B. Uppslagstabeller kan ha strängar som värden.
C. Uppslagstabeller kan ha listor som nycklar.
D. Uppslagstabeller kan ha listor som värden.
E. Uppslagstabeller kan vara tomma.

7. Vad skrivs ut av koden till höger?

A. `1`
B. `2`
C. `3`
D. `4`
E. `5`

```
x = 1

def f(x=2,y=3):
    print(x + y)

f(x)
```

8. Betrakta kodsnutten till höger, vad blir resultatet av `f([1,1,1])`?

A. 0
B. 3
C. 6
D. 9
E. Inget, då ett särfall lyfts eftersom man inte får ha två **`return`** i en funktion.

```
def f(mylist):
    if mylist == []:
        return 0
    else:
        return len(mylist) + f(mylist[1:])
```

Del B: kodfrågor (12p)

Var snäll använd ett papper till varje fråga i del B. Delfrågor får gärna lösas på samma papper.

9.

A. Skriv en funktion `add_positive(ns)` som tar in en lista `ns` med heltal (**`int`**) och returnerar summan av de positiva talen i listan. Negativa tal ska alltså ignoreras. (1p)

Exempelanvändning:

```
[In: ] print(add_positive([1,-2,3,-4]))
[Out:] 4
[In: ] print(add_positive([]))
[Out:] 0
```

B. Skriv ett program som frågar användaren efter heltal tills den skriver `0`. Då ska summan av de positiva talen summeras med hjälp av `add_positive` och resultatet skrivas ut. För full poäng måste man använda `add_positive` på lämpligt sätt. (2p)

Programmet ska även ha lämplig felhantering med hjälp av **`try`**/**`except`** så att det inte kraschar om användaren skriver in något som inte är ett tal. (1p)

Exempelkörning:

```
Write a number: 2
Write a number: -3
Write a number: 4
Write a number: hello
That is not a number!
Write a number: 7
Write a number: 0
Sum of positive numbers: 13
```

10. På den här uppgiften ska man skriva funktioner för att hantera inköpslistor.

A. En inköpslista kan modelleras som en lista av strängar, t.ex. `["Avocado","Milk","Butter"]`. Ett problem med detta är att man behöver repetera saker som ska köpas om man vill köpa flera av samma sort. Ett bättre sätt är att använda en uppslagstabell från strängar till heltal så att heltalen kan representera hur många som ska köpas av varje sak som ska handlas. Skriv en funktion `to_dict(shopping_list)` som tar in en lista `shopping_list` med strängar (eventuellt med dubbletter) och skapar en uppslagstabell. (2p)

Exempelanvändning:

```
[In: ] to_buy = ["Avocado","Avocado","Milk","Butter","Milk","Avocado"]
[In: ] print(to_dict(to_buy))
[Out:] {'Avocado': 3, 'Milk': 2, 'Butter': 1}
```

B. Det kan vara smidigt att ha sina inköpslistor i en fil. Skriv en funktion `dict_from_file(f)` som skapar en uppslagstabell från en fil `f` där första kolumnen är namnet på varan som ska handlas och andra kolumnen är antalet. (2p)

Obs: Koden ska fungera även om filen har samma vara på flera rader. Man kan anta att filen har rätt format och att antal saker som ska köpas alltid är ett positivt heltal.

Exempelfil `shoppinglist.txt` (observera att `Butter` förekommer på två ställen):

```
Avocado 3
Butter 1
Milk 2
Butter 2
Bread 3
```

Exempelanvändning:

```
[In: ] print(dict_from_file("shoppinglist.txt"))
[Out:] {'Avocado': 3, 'Butter': 3, 'Milk': 2, 'Bread': 3}
```

11. Inköpslistor, som i förra uppgiften, är ett exempel där objektorientering är smidigt för att modellera varor på ett bättre sätt än att bara använda strängar.

A. Skriv en klass `Item` med instansattributen `name` och `best_before`. Den första ska ha vara en sträng som innehåller namnet på varan. Den andra används för att spara bäst före datum, på formen `(y,m,d)` där `y` är år, `m` är månad och `d` är dag som varan går ut. Den ska alltså sparas som en tupel med tre heltal. Instansattributen ska kunna sättas med hjälp av en konstruktor. (1p)

B. Lägg till en `__str__` metod som skriver ut varans namn och bäst före datum inom parenteser (se exempel nedan). (1p)

C. Lägg även till en metod `is_still_ok(y,m,d)` som kollar så att varan inte har passerat sitt bäst före datum vid år `y`, månad `m` och dag `d`. Tänk först på hur datum jämförs innan ni löser den här uppgiften. (2p)

Exempelanvändning:

```
[In: ] b = Item("Banana",2026,3,20)
[In: ] print(b)
[Out:] Banana (best before: 20/3 2026)
[In: ] print(b.is_still_ok(2027,3,20))
[Out:] False
[In: ] print(b.is_still_ok(2026,4,2))
[Out:] False
[In: ] print(b.is_still_ok(2026,3,21))
[Out:] False
[In: ] print(b.is_still_ok(2026,3,19))
[Out:] True
[In: ] print(b.is_still_ok(2026,2,28))
[Out:] True
[In: ] print(b.is_still_ok(2025,12,31))
[Out:] True
```

Del C: kodfrågor inspirerade av labbarna (10p)

Var snäll använd ett papper till varje fråga i del C. Delfrågor får gärna lösas på samma papper.

12. Rankine är en mindre känd temperaturskala än de tre stora: Celsius, Fahrenheit och Kelvin. Tanken med Rankine är att den fungerar som Kelvin så att 0 Rankine är absoluta nollpunkten, men en grad Rankine motsvarar en grad Fahrenheit istället för en grad Celsius, så $R = 9/5K = 1.8K$.

A. För att konvertera en temperatur c Celsius till r Rankine används följande formel:

$$r = 1.8 * (c + 273)$$

För att gå åt andra hållet vänder man så klart bara på uträkningen:

$$c = r/1.8 - 273$$

Skriv funktionerna `celsius_to_rankine(t)` och `rankine_to_celsius(t)` som konverterar från Celsius till Rankine och från Rankine till Celsius. (1p)

Exempelanvändning:

```
[In: ] print(celsius_to_rankine(0))
[Out:] 491.40000000000003
[In: ] print(rankine_to_celsius(0))
[Out:] -273.0
```

Obs: Ingen avrundning behöver göras utan alla tal får vara oavrundade flyttal som i exemplet ovan.

B. Skriv ett program som ber användaren om vilken konvertering den vill göra (Celsius till Rankine eller Rankine till Celsius) och sedan frågar vilken temperatur som ska konverteras. Resultatet skrivs sedan ut till användaren. (2p)

Exempelkörning:

```
Which conversion do you want to do? Rankine to Celsius
Write temperature in Rankine to convert: 0
Temperature in Celsius: -273.0
```

13. På labb 3 skulle ni implementera insättningssortering. Nu ska ni implementera en annan känd sorteringsalgoritm: urvalssortering.

Algoritmbeskrivning för urvalssortering:

1. Skapa en tom lista för de sorterade elementen.
2. Hitta det minsta värdet i den osorterade listan, lägg det sist i den sorterade listan och ta bort det från den osorterade listan. Upprepa detta tills den osorterad listan är tom och alla element flyttats till rätt plats i den sorterade listan.
3. Returnera den sorterade listan.

Uppgifter:

A. Beskriv vilka steg algoritmen kommer ta för att sortera listan `[2,1,4,3]`. Dvs, förklara konkret hur algoritmen fungerar med detta indata-exempel. (1p)

B. Implementera `selection_sort(lst)` som implementerar algoritmen ovan för att sortera indatalistan `lst`. För poäng får man inte använda sig av någon inbyggd sorteringsfunktion. (2p)

Exempelanvändning:

```
[In: ] print(selection_sort([2,1,4,3]))
[Out:] [1,2,3,4]
```

14. Algoritmen given i uppgift 13 har nackdelen att man måste skapa en ny lista för de sorterade elementen. För att använda mindre datorminne är det istället bättre att formulera om algoritmen så att den jobbar ”in place”, dvs den skapar inte någon ny lista utan den byter bara plats på element i listan.

Algoritmbeskrivning för urvalssortering ”in place”:

1. Skapa en räknare, med startvärde `0`, som håller koll på hur många element som blivit sorterade. Tanken är att alla element med index mindre än räknaren är sorterade och de med index större än eller lika med räknaren är osorterade, på detta sätt behöver vi inte skapa någon ny lista utan kan göra allt med indatalistan.
2. Hitta det minsta värdet `m` i den osorterade delen av listan, dvs det minsta talet med index större än eller lika med räknaren. Byt ut `m` mot värdet på det index som räknaren pekar på. Öka räknaren med ett då ett till element blivit sorterat.
3. Repetera steg 2 till alla element är sorterade, dvs till räknaren är lika med längden på listan.

Uppgifter:

A. Beskriv vilka steg algoritmen kommer ta för att sortera listan `[2,1,4,3]`. (1p)

B. Skriv `selection_sort_in_place(lst)` som implementerar algoritmen ovan. För poäng får man inte använda sig av någon inbyggd sorteringsfunktion. (3p)

Exempelanvändning:

```
[In: ] print(selection_sort_in_place([2,1,4,3]))
[Out:] [1,2,3,4]
```

English translation

- The exam has multiple-choice questions where at least one answer option is correct. If you answer incorrectly or do not select the exact number of correct alternatives, you receive zero points for the question.
- Part A consists of multiple-choice questions (a total of 8 points).
- Part B consists of coding questions with varying point values (a total of 12 points).
- Part C consists of coding questions inspired by the labs, with varying point values (a total of 10 points).
- No import (Python's standard library or external libraries) may be used unless they are mentioned or included in the assignment. You may use built-in functions such as len, range, and map.
- All code refers to **Python 3**, i.e., *not* e.g. Python 2.7.
- **Allowed aids:** One A4 sheet with as much information as you want. You may write on both sides.
- **Grade thresholds:** E: 15, D: 18, C: 21, B: 24, A: 27, out of a maximum of 30.

Part A: multiple-choice questions (8 points)

Please gather your answers to part A on a single answer sheet. Do not mark your answers on this paper!

1. Consider the code snippet on the right; what is the value of `x` after the code has been executed?

A. 2
B. 4
C. 6
D. 8
E. 10

```
s = "20260314"
x = 0
for c in s:
    d = int(c)
    if d % 2 == 0:
        x += d
    else:
        x -= d
```

2. Which of the following assignments make `x and (not y and z)` evaluate to `False`?

A. `x = True`, `y = False`, och `z = True`
B. `x = False`, `y = False`, och `z = True`
C. `x = True`, `y = True`, och `z = True`
D. `x = False`, `y = False`, och `z = False`
E. `x = False`, `y = True`, och `z = True`

3. Consider the code snippet on the right; what is the value of `out` after it has been executed?

A. `Python`
B. `Pyytttthhhhooooonnnnnn`
C. `ytthhhoooonnnnn`
D. `nnnnnnooooohhhhtttyyP`
E. `nnnnnoooohhhtty`

```
s = "Python"
out = ""
i = 0

while i < len(s):
    out += i * s[i]
    i += 1
```

4. What is **`break`** used for in Python?

A. Break a string in the middle so that you can continue it on the next line.
B. Terminate a loop.
C. Raise the special case `ProgramBreakError`.
D. Shut down Spyder.
E. There is no such thing as **`break`** in Python.

5. Consider the code snippet on the right; what is the value of `s` after it has been executed?

A. 3

B. 6

C. 8

D. None, since an exception is raised because you cannot have lists in dictionaries.

E. None, since an exception is raised because you cannot compute the length of a dictionary.

```
d = { 2 : "hej",
      3 : [1,2,3],
      4 : { 5 : "hej" , 6 : "du" } }

s = 0

for x in d:
    s += len(d[x])
```

6. Which of the following statements are true about Python?

A. Lookup tables can have strings as keys.

B. Lookup tables can have strings as values.

C. Lookup tables can have lists as keys.

D. Lookup tables can have lists as values.

E. Lookup tables can be empty.

7. What is printed by the code on the right?

A. `1`

B. `2`

C. `3`

D. `4`

E. `5`

```
x = 1

def f(x=2,y=3):
    print(x + y)

f(x)
```

8. Consider the code snippet on the right. What is the result of `f([1,1,1])`?

A. 0

B. 3

C. 6

D. 9

E. Nothing, since an exception is raised because you cannot have two **`return`** in a function.

```
def f(mylist):
    if mylist == []:
        return 0
    else:
        return len(mylist) + f(mylist[1:])
```

Part B: code problems (12 points)

Please use a paper for each problem in part B. You are welcome to use the same sheet for multiple subproblems. Only write on one side!

9.

A. Write a function `add_positive(ns)` that takes a list `ns` with integers (**`int`**) as input and returns the sum of the positive numbers in the list. Negative numbers should be ignored. (1p)

Example usage:

```
[In: ] print(add_positive([1,-2,3,-4]))
[Out:] 4
[In: ] print(add_positive([]))
[Out:] 0
```

B. Write a program that asks the user for integers until they enter `0`. Then the sum of the positive numbers should be computed using `add_positive`, and the result should be printed. For full points, `add_positive` must be used in an appropriate way. (2p)

The program should also include appropriate error handling using **`try`**/**`except`** so that it does not crash if the user enters something that is not a number. (1p)

Example:

```
Write a number: 2
Write a number: -3
Write a number: 4
Write a number: hello
That is not a number!
Write a number: 7
Write a number: 0
Sum of positive numbers: 13
```

10. In this assignment we look at functions for managing shopping lists.

A. A shopping list can be modeled as a list of strings, e.g. `["Avocado","Milk","Butter"]`. One problem with this is that you need to repeat items that should be bought if you want to buy several of the same kind. A better approach is to use a lookup table from strings to integers so that the integers can represent how many of each item should be purchased. Write a function `to_dict(shopping_list)` that takes a list `shopping_list` of strings (possibly with duplicates) and creates a lookup table. (2p)

Example usage:

```
[In: ] to_buy = ["Avocado","Avocado","Milk","Butter","Milk","Avocado"]
[In: ] print(to_dict(to_buy))
[Out:] {'Avocado': 3, 'Milk': 2, 'Butter': 1}
```

B. It can be convenient to keep your shopping lists in a file. Write a function `dict_from_file(f)` that creates a lookup table from a file `f` where the first column is the name of the item to be purchased and the second column is the quantity. (2p)

Note: The code should work even if the file contains the same item on multiple lines. You may assume that the file has the correct format and that the quantities to be purchased are always positive integers.

Example file `shoppinglist.txt` (note that `Butter` appears in two places):

```
Avocado 3
Butter 1
Milk 2
Butter 2
Bread 3
```

Example usage:

```
[In: ] print(dict_from_file("shoppinglist.txt"))
[Out:] {'Avocado': 3, 'Butter': 3, 'Milk': 2, 'Bread': 3}
```

11. Shopping lists, as in the previous exercise, are an example where object orientation is convenient for modeling items in a better way than by using only strings.

A. Write a class `Item` with the instance attributes `name` and `best_before`. The first should be a string containing the name of the item. The second is used to store the best-before date in the form `(y,m,d)`, where `y` is the year, `m` is the month, and `d` is the day the item expires. It should therefore be stored as a tuple with three integers. The instance attributes should be set using a constructor. (1p)

B. Add a `__str__` method that prints the item's name and its best-before date in parentheses (see the example below). (1p)

C. Also add a method `is_still_ok(y,m,d)` that checks that the item has not passed its best-before date as of year `y`, month `m`, and day `d`. Think first about how dates are compared before solving this task. (2p)

Example usage:

```
[In: ] b = Item("Banana",2026,3,20)
[In: ] print(b)
[Out:] Banana (best before: 20/3 2026)
[In: ] print(b.is_still_ok(2027,3,20))
[Out:] False
[In: ] print(b.is_still_ok(2026,4,2))
[Out:] False
[In: ] print(b.is_still_ok(2026,3,21))
[Out:] False
[In: ] print(b.is_still_ok(2026,3,19))
[Out:] True
[In: ] print(b.is_still_ok(2026,2,28))
[Out:] True
[In: ] print(b.is_still_ok(2025,12,31))
[Out:] True
```

Part C: coding problems inspired by the labs (10 points)

12. Rankine is a less well-known temperature scale than the three major ones: Celsius, Fahrenheit, and Kelvin. The idea behind Rankine is that it works like Kelvin so that 0 Rankine is absolute zero, but one degree Rankine corresponds to one degree Fahrenheit instead of one degree Celsius, so $R = \frac{9}{5}K = 1.8\,K$.

A. To convert a temperature c in Celsius to r in Rankine, the following formula is used:

$$r = 1.8 * (c + 273)$$

To go the other way, you simply reverse the calculation:

$$c = r/1.8 - 273$$

Write the functions `celsius_to_rankine(t)` and `rankine_to_celsius(t)` that convert from Celsius to Rankine and from Rankine to Celsius. (1p)

Example usage:

```
[In: ] print(celsius_to_rankine(0))
[Out:] 491.40000000000003
[In: ] print(rankine_to_celsius(0))
[Out:] -273.0
```

Note: No rounding is required; all numbers may remain unrounded floating-point values as in the example above.

B. Write a program that asks the user which conversion they want to perform (Celsius to Rankine or Rankine to Celsius) and then asks which temperature should be converted. The result is then printed to the user. (2p)

Example run:

```
Which conversion do you want to do? Rankine to Celsius
Write temperature in Rankine to convert: 0
Temperature in Celsius: -273.0
```

13. In lab 3 you were asked to implement insertion sort. Now you will implement another well-known sorting algorithm: selection sort.

Algorithm description for selection sort:

1. Create an empty list for the sorted elements.
2. Find the smallest value in the unsorted list, place it last in the sorted list, and remove it from the unsorted list. Repeat this until the unsorted list is empty and all elements have been moved to the correct position in the sorted list.
3. Return the sorted list.

Tasks:

A. Describe the steps the algorithm will take to sort the list `[2,1,4,3]`. That is, explain concretely how the algorithm works with this input example. (1p)

B. Implement `selection_sort(lst)` that implements the algorithm above to sort the input list `lst`. To receive points, you may not use any built-in sorting function. (2p)

Example usage:

```
[In: ] print(selection_sort([2,1,4,3]))
[Out:] [1,2,3,4]
```

14. The algorithm given in problem 13 has the disadvantage that you must create a new list for the sorted elements. To use less memory, it is instead better to reformulate the algorithm so that it works "in place", i.e., it does not create any new list but only swaps elements within the list.

Algorithm description for in-place selection sort:

1. Create a counter, starting at `0`, that keeps track of how many elements have been sorted. The idea is that all elements with an index smaller than the counter are sorted, and those with an index greater than or equal to the counter are unsorted. In this way, we do not need to create a new list but can do everything using the input list.
2. Find the smallest value `m` in the unsorted part of the list, i.e., the smallest number with an index greater than or equal to the counter. Swap `m` with the value at the index indicated by the counter. Increase the counter by one since one more element has now been sorted.
3. Repeat step 2 until all elements are sorted, i.e., until the counter is equal to the length of the list.

Tasks:

A. Describe the steps the algorithm will take to sort the list `[2,1,4,3]`. (1p)

B. Write `selection_sort_in_place(lst)` that implements the algorithm above. To receive points, you may not use any built-in sorting function. (3p)

Example usage:

```
[In: ] print(selection_sort_in_place([2,1,4,3]))
[Out:] [1,2,3,4]
```