TDDE44. DAT1. 2hp.

2025-01-08 kl. 14.00-18.00

Tillåtna hjälpmedel: penna, papper, linjal, suddgummi, godkänd(a) bok/böcker (lärare kontrollerar böcker innan tentan)

Givna filer

Du hittar eventuella filer du kan behöva till uppgifterna i katalogen `~/Desktop/given_files`. Denna katalog är **skrivskyddad**, liksom filerna i den. Behöver du ändra i någon av de givna filerna måste du först kopiera filen till en annan katalog.

Skriva kod

Det rekommenderas att du sparar dina lösningar på skrivbordet. Sökvägen till skrivbordet är `~/Desktop`. Spara din kod för varje uppgift inklusive deluppgifter i *en* fil. Du kan *inte* spara din fil i `~/Desktop/given_files`. Exempel:

- Svaren till uppgift 1.1 (1.1.1, 1.1.2) sparas i filen (`uppg11.py`).
- Svaret till uppgift 1.2 sparas i filen (`uppg12.py`).

Inga doc-strängar behöver skrivas. Inga avdrag kommer ges om koden bryter mot `PEP8` eller `PEP257`.

Begränsningar av hur en uppgift får lösas

I de fall som en uppgift ska lösas på ett speciellt sätt står detta angivet i uppgiften; t.ex. "*Lös uppgiften rekursivt.*" eller "*Lös uppgiften med hjälp av en `for`-loop*".

Om specifika funktioner inte får användas i lösningen står detta också angivet i uppgiften, t.ex. "*Funktionen `max()` får inte användas.*"

Tillåtna moduler att använda på tentan: Följande Python-moduler är *tillåtna* att användas.

- `copy`
- `csv`
- `math`
- `random`
- `sys`

Övriga moduler är otillåtna om inget annat anges.

Notation och termer i uppgifter

När strängar visas i uppgifterna kommer de alltid att vara omgivna av citattecken. Om det står att strängen `"hej"` ska skrivas ut, ska utskriften motsvara `print("hej")`.

- termen *heltal* syftar alltid på datatypen `int` om inget annat anges
- termen *sträng* syftar alltid på datatypen `str` om inget annat anges

Exempel i uppgifter

Tänk på att eventuella exempel till uppgifterna *inte täcker alla möjliga fall*. Ibland kan t.ex. divisioner resultera i svar med precisionsfel. Om du får `1.99999999` som svar vid divisionen `4/2.0` behöver du inte oroa dig.

Kom ihåg att testköra din kod

Inkompletta och delvis felaktiga lösningar *kan* ge delpoäng men kom ihåg att testköra din kod innan du skickar in! **Filer som kraschar vid körning ger normalt sett underkänt/0p på hela uppgiften.** Alltså, om du vill skicka in inkompletta lösningar som kraschar när du testkör så **kommentera ut eventuella anrop till den kraschande koden innan du skickar in**.

Rättning/bedömning av Del 1

Del 1 består av fem uppgifter. För att få godkänt på Del 1 måste **alla uppgifter** (1.1-1.5) vara godkända.

- För att få **godkänt** på en uppgift (t.ex. 1.1) måste samtliga deluppgifter vara godkända.
- Ej godkända uppgifter kan kompletteras i mån av tid.
- *Upprepade inlämningar där du uppenbart gissat dig fram kan komma att underkännas och kan då inte längre kompletteras.* Om du inte förstår feedbacken så be om ett förtydligande istället för att gissa.

Rättning/bedömning av Del 2

Del 2 rättas endast om Del 1 är godkänd. Om du inte fått godkänt på Del 1 men vill få kommentarer på Del 2 kan du kontakta examinator efter att du fått ditt resultat.

Del 2 består av tre frågor som är något mer omfattande och som kräver mer problemlösning än uppgifterna i Del 1.

Du kan få 0-5 poäng per uppgift i Del 2.

Betyg på tentan

- Betyg 3: Godkänd på Del 1
- Betyg 4: Godkänd på Del 1 + 5 poäng på Del 2.
- Betyg 5: Godkänd på Del 1 + 10 poäng på Del 2.

Del 1

För att bli godkänd på Del 1 måste alla uppgifter vara godkända.

Uppgift 1 - Loopar

- För att få godkänt på denna uppgift skall **båda** deluppgifterna lösas.
- En av uppgifterna skall lösas med en `while`-loop och en skall lösas med `for`-loop.
- Comprehensions och generatorer får *inte* användas för att lösa uppgifterna.
- Spara lösningarna till båda deluppgifterna i Uppgift 1 i en fil med namnet `uppg1.py`.
- Filen skickas in via Studentklienten som '*Assignment #1*'.

Uppgift 1.1 - lös med lämplig loop

Skriv funktionen `get_next_power_of_two(val)` vars argument är ett heltal större än 0. Funktionen ska returnera närmsta tvåpotens (2^x) som är *större* än `val`.

Uppgiften skall lösas med en lämplig loop. Dvs, du får t.ex. inte använda logaritmer eller metoden `bit_length()` för att räkna ut lösningen.

Exempel

```
>>> get_next_power_of_two(70)
128  # 2^7
>>> get_next_power_of_two(8)
16  # 2^4
```

Uppgift 1.2 - lös med lämplig loop

Skriv funktionen `count_swedish_vowels(word)` vars argument `word` är en sträng. Antag att alla bokstäver i `word` är gemener. Funktionen ska returnera hur många av bokstäverna i `word` som är svenska vokaler. Följande bokstäver är vokaler: aouåeiyäö.

Uppgiften skall lösas med en lämplig loop.

Tips: Operatorn `in` kan användas på strängar; `'a' in 'abc'` → `True`.

Exempel

```
>>> count_swedish_vowels('bokomslag')
3
>>> count_swedish_vowels('pythonprogram')
4
>>> count_swedish_vowels('hmm')
0
```

Uppgift 2 - Rekursion

- Spara lösningen till Uppgift 2 i en fil med namnet `uppg2.py`.
- Filen skickas in via Studentklienten som '*Assignment #2*'.

Skriv funktionen `count_occurrences_rec(values, target)` som får in en lista av strängar och tal. Funktionen ska returnera antalet element i `values` som har värdet `target`. Svaret som returneras ska vara ett heltal.

Notera att `target` endast ska matcha hela element i `values`. Se t.ex. första exemplet nedan där inga fall av strängen `'e'` hittas eftersom `'e'` endast förekommer som en delsträng i `'hej'`.

Uppgiften skall lösas med rekursion och det är inte tillåtet att använda t.ex. list-metoden `count`.

Exempel

```
>>> count_occurrences_rec([1, 2, 3, 3, 'hej', 2, 'h', 0.4, 2, '2'], 'e')
0
>>> count_occurrences_rec([1, 2, 3, 3, 'hej', 2, 'h', 0.4, 2, '2'], '2')
1
>>> count_occurrences_rec([1, 2, 3, 3, 'hej', 2, 'h', 0.4, 2, '2'], 2)
3
```

Uppgift 3 - Dictionaries och nästling

- Spara lösningen till Uppgift 3 i en fil med namnet `uppg3.py`.
- Filen skickas in via Studentklienten som '*Assignment #3*'.

Ett dictionary används för att lagra information om en läsgrupps lästa böcker. Detta dictionary innehåller boktitlar som nycklar vilka har varsitt inre dictionary som värde. Dessa inre dictionaries innehåller antalet sidor boken har under nyckeln `'pages'` samt vilka personer som läst boken under nyckeln `'read_by'`. Exempelvis:

```
cs_reading_group = {
    'The Wizard Book': {
        'pages': 657, 'read_by': ['Alan', 'Grace']
    },
    'The Dragon Book': {
        'pages': 1040, 'read_by': ['Margaret', 'Grace']
    },
    'The Dinosaur Book': {
        'pages': 896, 'read_by': ['John', 'Alan', 'Margaret']
    },
    'The Cinderella Book': {
        'pages': 500, 'read_by': ['Alan', 'Grace', 'John']
    }
}
```

(Om du av något skäl inte kan kopiera från denna pdf så finns samma dictionary i den givna filen `cs_reading_group.py`.)

Skriv funktionen `total_pages_read(reading_log, person)` som givet ett dictionary enligt ovan returnerar det totala antalet sidor som en person har läst. Om personen inte läst några av böckerna returneras 0 (se sista exemplet nedan).

Exempel Givet `'cs_reading_group'` enligt ovan:

```
>>> total_pages_read(cs_reading_group, 'Alan')
2053
>>> total_pages_read(cs_reading_group, 'Margaret')
1936
>>> total_pages_read(cs_reading_group, 'Daniel')
0
```

Uppgift 4

- Spara lösningen till Uppgift 4 i en fil med namnet `uppg4.py`.
- Filen skickas in via Studentklienten som '*Assignment #4*'.

I denna uppgift ska klassen `CheckingAccount` implementeras som ska användas i ett bankprogram. Klassen ska ha instansvariabeln `transactions` som implementeras som en lista (varje element i den listan är en transaktion som gjorts). Implementera klassen enligt nedan:

- När en instans skapas av klassen anger man hur mycket pengar som finns från början, t.ex. `CheckingAccount(500)`. Beloppet sparas som den första positiva transaktionen i instansvariabeln `transactions`.
- Klassen ska ha metoden `get_balance()` som ska returnera hur mycket pengar som finns på kontot, dvs resultatet av alla transaktioner.
- Klassen ska ha metoden `add_transaction(amount)` som används för att bokföra en transaktion som läggs till listan över transaktioner. Om `amount` är positivt har man satt in pengar. Om `amount` är negativt har man hämtat ut pengar.

Exempel

```
>>> account = CheckingAccount(500)
>>> account.get_balance()
500
>>> account.add_transaction(207)
>>> account.add_transaction(-206)
>>> account.get_balance()
501
>>> account.add_transaction(-303)
>>> account.add_transaction(-115)
>>> account.add_transaction(614)
>>> account.get_balance()
697
>>> account.transactions
[500, 207, -206, -303, -115, 614]
```

Uppgift 5

- Spara lösningen till Uppgift 5 i en fil med namnet `uppg5.py`.

Filen `adresser.csv` hör ihop med denna uppgift.

Vi har lagrat ett antal adresser i en fil där varje rad är en adress på följande format (ingen rubrikrad finns i filen):

`förnamn;efternamn;gatunamn;husnummer;postnummer;postort`

Din uppgift är att skriva funktionen `show_people_on_street(filepath, streetname)` som tar in två strängar som argument. Argumentet `filepath` är sökvägen till en fil som innehåller adresser enligt formatet ovan. Argumentet `streetname` är namnet på en gata (utan husnummer).

Funktionen ska läsa in data från den angivna filen och *skriva ut* vilka som bor på gatan `streetname` enligt följande format.

```
Det bor X personer(er) på Y-gatan:
Förnamn Efternamn
Förnamn Efternamn
...
```

Om det inte finns någon som bor på de eftersökta gatunamnet ska funktionen *skriva ut* 'Inga personer bor på Y-gatan.' (ersätt Y-gatan med rätt gatunamn). Dela upp din lösning i fler funktioner efter behov.

Exempel

```
>>> show_people_on_street('adresser.csv', 'Engatan')
Det bor 2 person(er) på Engatan:
Elise Ekberg
Einar Ek
>>> show_people_on_street('adresser.csv', 'Tvågatan')
Inga personer bor på Tvågatan.
```

Del 2

- För att få betyg 4 krävs 5 poäng från denna del.
- För att få betyg 5 krävs 10 poäng från denna del.

Dela upp dina lösningar i flera funktioner efter behov.

Uppgift 6 (5p)

- Spara svaret till Uppgift 6 i en fil med namnet `uppg6.py`.
- Filen skickas in via Studentklienten som '*Assignment #6*'.

Vi har en skattkista som består av lådor som sitter ihop i en rad. Varje låda skyddas av ett kraftfält och innehåller en juvel. Om kraftfältet är av så kommer vi åt juvelen i lådan. Vi representerar skattkistans lådor som en lista:

```
[0, 1, 1, 0]
```

Värdet `0` betyder att kraftfältet är avstängt och `1` betyder att kraftfältet är på. När vi plockar ut en juvel från en låda med avstängt kraftfält, så känner lådan av det och alla avstängda kraftfält slås på samtidigt som alla kraftfält som var på stängs av. Av anledningar som inte är relevanta för uppgiften kan vi bara plocka juveler i sekvens (från vänster till höger).

Om vi plockar juvelen från första lådan ovan resulterar det i följande tillstånd

```
[1, 0, 0, 1]
```

Vi kan nu, plocka juvelen från andra lådan (index 1) och tillståndet ändras till

```
[0, 1, 1, 0]
```

Vi kan plocka juvelen från sista lådan (index 3) och har alltså fått totalt tre juveler.

Din uppgift är att skriva funktionen `gather_jewels(treasure_boxes)` som får en lista enligt ovan och ska returnera antalet juveler som plockas.

Exempel

```
>>> gather_jewels([0, 1, 1, 0])
3
>>> gather_jewels([0, 0, 1, 1, 1, 0, 1])
4
>>> gather_jewels([0, 0, 0])
1
>>> gather_jewels([1, 1, 1])
0
>>> gather_jewels([1, 1, 0])
1
```

Uppgift 7 (5p)

- Spara svaret till Uppgift 7 i en fil med namnet `uppg7.py`.
- Filen skickas in via Studentklienten som '*Assignment #7*'.

Heltalet `121314` kan delas in i grupper om två vilket ger oss sekvensen heltal `12`, `13`, `14`. I denna sekvens följs varje heltal av det nästa heltalet i stigande ordning.

Heltalet `123122` kan delas in i grupper om tre som ger oss en sekvens heltal i sjunkande ordning: `123`, `122`.

Din uppgift är att skriva funktionen `is_valid_sequence(number)` som returnerar `True` om heltalet `number` kan delas in i grupper av *någon* storlek så att heltalen i sekvensen är antingen i *stigande ordning*, eller i *sjunkande ordning*. Om detta inte går, ska funktionen returnera `False`

Sekvensen måste vara på minst två tal och uppdelningen ska gå jämt upp.

Exempel

```
>>> is_valid_sequence(123) # 1 2 3 är en stigande sekvens
True
>>> is_valid_sequence(8910) # 8 9 10 är inte godkänd; grupperna har inte lika många siffro
False
>>> is_valid_sequence(999897) # 99 98 97 är en sjunkande sekvens
True
>>> is_valid_sequence(12012112)
False
>>> is_valid_sequence(120121122) # 120 121 122 är en stigande sekvens
True
```

Uppgift 8 (5p)

- Spara svaret till Uppgift 8 i en fil med namnet `uppg8.py`.
- Filen skickas in via Studentklienten som '*Assignment #8*'.

Vi har en bokdatabas på 55 böcker. I filen `book_ratings.txt` finns data för hur olika personer betygsatt olika böcker. Varannan rad med början på första raden innehåller namnet på en person. Efterföljande rad innehåller en sekvens av 55 heltal (åtskilda av ett mellanslag) som är personens betyg för de 55 böcker som finns med i databasen. Heltalen har följande betydelse:

- `-5`: Hatade boken.
- `-3`: Gillade inte boken.
- `0`: Har inte läst boken.
- `1`: Boken var OK.
- `3`: Gillade boken.
- `5`: Gillade boken jättemycket!

Vi kan se sekvensen av de 55 betygen som en vektor med 55 dimensioner. Som ett mått på likhet mellan två vektorer använder man ofta cosinus för vinkeln mellan dem, $\cos\theta$. Ett värde av $\cos\theta$ nära 1 kan tolkas som att personerna tycker relativt lika och ett värde nära -1 kan tolkas som att de tycker relativt motsatt varandra.

Din uppgift är att skriva funktionen `cosine_similarity(name1, name2, filepath)` som får in två namn och en sökväg till en fil med bokbetyg enligt ovan. Funktionen ska returnera hur lika personerna med namnen `name1` och `name2` betygsatt böckerna i databasen. D.v.s. funktionen ska läsa in de två personernas betygsättningar som vektorer och returnera $\cos\theta$ mellan dem.

Det är högst lämpligt att dela upp sin lösning i flera funktioner.

Det är inte tillåtet att använda externa moduler som `SciPy` eller `NumPy` utan endast de vanliga matematiska operatorerna och modulen `math` får användas för matematiken. Finurlig tillämpning av funktioner som `sum` och `zip` kan vara användbart och ge högre poäng men är inte nödvändigt för att lösa uppgiften.

En minimal formelsamling för den nödvändiga linjära algebran följer efter exempelkörningarna på nästa sida.

Exempel

```
>>> cosine_similarity('Tony', 'Tony', 'book_ratings.txt')
1.0000000000000002
>>> cosine_similarity('Tony', 'joe', 'book_ratings.txt')
0.06422343987720157
>>> cosine_similarity('NuNu', 'Megan', 'book_ratings.txt')
0.5044088467483089
>>> cosine_similarity('McLovin', 'Boxxy', 'book_ratings.txt')
-0.4745789978762495
```

Formelsamling

En formel för $\cos\theta$ mellan två vektorer a och b kan härledas ur definitionen av skalärprodukten:

$$\mathbf{a} \cdot \mathbf{b} = |\mathbf{a}|\,|\mathbf{b}| \cos\theta \iff \frac{\mathbf{a} \cdot \mathbf{b}}{|\mathbf{a}|\,|\mathbf{b}|} = \cos\theta$$

där $\mathbf{a} \cdot \mathbf{b}$ är skalärprodukten mellan $\mathbf{a}$ och $\mathbf{b}$, och $|\mathbf{a}|\,|\mathbf{b}|$ är produkten av vektorernas längd.

Skalärprodukten mellan två vektorer, $\mathbf{a} \cdot \mathbf{b}$, där $\mathbf{a} = (a_1, a_2, ...a_n)$ och $\mathbf{b} = (b_1, b_2, ...b_n)$ kan beräknas som

$$\sum_{i=1}^{n} a_i \cdot b_i$$

T.ex. $(1, 2, 3) \cdot (4, 5, 6) = 1 \cdot 4 + 2 \cdot 5 + 3 \cdot 6 = 32$

Längden av en vektor syftar här på den Euklidiska normen som kan beräknas enligt formeln

$$|\mathbf{a}| = \sqrt{\sum_{i=1}^{n} a_i^2}$$

(Notera att ovanstående är lika med kvadratroten ur en vektors skalärprodukt med sig själv.)