TDDE44. Extra tentaövningsuppgifter.

Uppdaterade maj 2021

Tenta-ID: TDDE44-DAT1-Ovningsuppgifter

Tillåtna hjälpmedel: Det är endast tillåtet att använda kurslitteratur (se Kursinformation på kurshemsidan) som hjälpmedel under tentan, samt Kapitel 2 och 4 från pythondokumentationen (länk nedan). Det är alltså **inte tillåtet** att t.ex. söka på nätet, referera till föreläsningsbilder eller uppgifter ni gjort under kursen.

- Kapitel 2 och 4 från dokumentationen till Python 3.6.12

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

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

Skriva kod

- Inga doc-strängar behöver skrivas. Inga avdrag kommer ges om koden bryter mot `PEP8` eller `PEP257`.
- Kommentarer är inte nödvändiga, men *kan* t.ex. för uppgifter i Del 2 bidra till poäng om de visar att du tänkt rätt, men missat i implementationen.

Begränsningar av hur en uppgift får lösas

Vissa uppgifter kan ha begränsningar för hur deluppgifterna får lösas. Dessa står i anslutning till respektive uppgift.

Övrigt

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.

Del 1

- *List comprehensions* får inte användas för att lösa uppgifterna i Del 1

Loopar (1 poäng per uppgift)

Öva på att lösa med både `for`- och `while`-loop, samt rekursion. Vissa uppgifter går bara att lösa med en viss typ av loop, eller kan kräva att man skickar med extra argument för att lösa med rekursion.

get_max_int (1p)

Skriv funktionen `get_max_int(value_list)` som ska returnera det högsta heltalet (datatypen `int`) i listan `value_list` som är en lista med element av datatyperna `int` och `str` (en lista med heltal och strängar).

OBS! Även negativa heltal är heltal.

Exempel

- Anropet `get_max_int([-9, 3, 6, "a", 3, 100, 2, 30]` ska returnera `100`.
- Anropet `get_max_int(["a", -9, -8, -100]` ska returnera `-8`.

get_unique_random_integers (1p)

Skriv funktionen `get_unique_random_integers(start, stop, number_of_ints)` vars argument är alla heltal större än 0 och `start` $<$ `stop`. Funktionen ska returnera en lista med som innehåller slumpmässiga heltal mellan `start` och `stop` (inklusive `start` och `stop`). Inget av heltalen får förekomma mer än en gång. Antag också att `number_of_ints` $<$ `stop-start`.

Använd funktionen `randint(a, b)` från modulen `random` som returnerar ett slumpmässigt heltal `c`, där `a` $\leqslant$ `c` $\leqslant$ `b`.

Exempel

- Anropet `get_unique_random_integers(1, 100, 10)` returnerar en lista med 10 heltal, t.ex. `[93, 23, 6, 26, 90, 29, 59, 12, 15, 86]`.

remove_all_strings (1p)

Skriv funktionen `remove_all_strings(values)` som ska returnera en lista som innehåller alla värden i `values` som inte är strängar. `values` är en lista med noll eller fler element och kan innehålla strängar, heltal och flyttal.

Exempel

- `remove_all_strings([])` returnerar `[]`
- `remove_all_strings(["hejsan"])` returnerar `[]`
- `remove_all_strings(["hejsan", 1, 0.5, "hopp"])` returnerar `[1, 0.5]`

Dictionaries och nästlade strukturer (1 poäng per uppgift)

get_popular_words (1p)

Skriv funktionen `get_popular_words(word_freqs)` som tar in ett dictionary vars nycklar är ord (datatyp `str`) och vars värden är antalet förekomster av det ordet i en text (datatyp `int`). Funktionen ska returnera en lista med alla ord som förekommer fler än 50 gånger.

Exempel

```
wf = { 'hej': 1, 'och': 4, 'förlåt': 204, 'havregröt': 50, 'träd': 102 }
```

Givet ovanstående ska `get_popular_words(wf)` returnera `['förlåt', 'träd']`

count_pets (1p)

Skriv funktionen `count_pets(people)` ska returnera det totala antalet djur som människorna i listan `people` äger tillsammans. Varje element i listan `people` är ett dictionary med nycklarna `"name"`, `"age"` och `"number of pets"`. Värdet tillhörande nyckeln `"name"` är personens namn. Värdet tillhörande nyckeln `"age"` är personens ålder, värdet tillhörande nyckeln `"number of pets"` är antalet husdjur som personen äger. Exemplet nedan är en lista som innehåller tre personer:

```
my_friends = [ { "name": "Ada", "age": 5, "number of pets": 2 },
               { "name": "Bertil", "age": 8, "number of pets": 0 },
               { "name": "Celeste", "age": 32, "number of pets": 4 } ]
```

Exempel Givet ovanstående värde på `my_friends` ska anropet `count_pets(my_friends)` returnera `6`.

Definiera klass (1 poäng per metod)

class_coinwallet (3p)

I denna uppgift ska du implementera klassen `CoinWallet` som ska användas som en plånbok. Klassen ska definiera en instansvariabel `coins` som implementeras som en lista. Implementera klassen enligt nedan:

- När en instans skapas av klassen måste man ange vilka mynt som finns i den från början. Man ska alltså kunna skriva `CoinWallet([10, 10, 10, 5, 5])` för att skapa en instans som innehåller tre 10-kronorsmynt och två 5-kronorsmynt.
- Klassen ska ha metoden `get_total()` som ska returnera hur mycket pengar som finns. Om innehållet är tre 10-kronorsmynt och två 5-kronorsmynt så ska `40` returneras.
- Klassen ska ha metoden `add_coin(coin_value)` som tar emot ett heltal och lägger till det till instansvariabeln `coins`.

class_book (3p)

I denna uppgift ska du definiera klassen `Book`. Vi vill lagra bokens titel, vem som är författare till boken, vilket år den kom ut samt godtyckligt antal taggar/etiketter för boken. Implementera klassen enligt nedan:

- När en instans av klassen skapas måste titel (`str`), författare (`str`) och årtal (`int`) skickas med som argument, t.ex. `Book("Learning Python", "Mark Lutz", 2013)`. Spara titeln i instansvariabeln `title`, författaren i instansvariabeln `author` och året i instansvariabeln `year`. Instansvariabeln `tags` som vi sedan ska använda för att lagra bokens taggar sätts till en tom lista.
- Implementera metoden `add_tag(tag)` som lägger till strängen `tag` till instansvariabeln `tags`. Om `tag` redan finns i instansvariabeln `tags` ska den *inte* läggas till igen.
- Implementera metoden `__str__()` som returnerar en beskrivning av av boken som en sträng på formatet `"<title>. <author>. (<year>) tags: <tags>"`. Se exempel nedan.

Exempel

```
>>> book = Book("Learning Python", "Mark Lutz", 2013)
>>> book.tags
[]
>>> print(book)
Learning Python. Mark Lutz. (2013) tags:
>>> book.add_tag("programming")
>>> book.add_tag("python")
>>> print(book)
Learning Python. Mark Lutz. (2013) tags: programming, python
```

Del 2 (enklare uppgifter)

robberlanguage (1p)

Skriv funktionen `robberlanguage(text)` som tar in en sträng och returnerar strängen omgjord till rövarspråket.

Regeln för rövarspråket är att man efter varje konsonant lägger till ett `"o"` och därefter samma konsonant igen, till exempel blir `"b"` till `"bob"` och `"f"` blir `"fof"`. Vokaler förblir oförändrade (svenska vokaler: aouåeiyäö). `"Jag"` blir alltså `"Jojagog"`.

Din implementationen av funktionen ska inte ändra på skiljetecken och mellanslag. Versaler (stora bokstäver) ska bibehållas. Om en konsonant är versal ska den bli till en sekvens där den första förekomsten är versal, men den andra förekomsten är gemen (liten). `"Hej!"` blir alltså `"Hohejoj!` och `"Ojdå!"` blir `"Ojojdodå!"`

Undantag

- `"c"` kan uttalas både som `"s"` och `"k"` beroende på sammang. I din implmentation räcker det med att du behandlar teckensekvensen `"ck"` som `"kk"`. Strängen `"tack"` ska alltså bli `"totakokkok"` medan strängen `"cykel"` blir `"cocykokelol"`.
- `"x"` betraktas i rövarspråket som om det skrevs som `"ks"`. Strängen `"sax"` ska därför bli `"sosakoksos"`.

Exempel

- Anropet `robberlanguage("Hej Xerxes!")` returnerar `"Hohejoj Koksoserorkoksosesos!"`
- Anropet `robberlanguage("Oj, backa!")` returnerar `"Ojoj, bobakokkoka!"`

caesar_decoded (1p)

Julius Caesar sägs ha kodat hemliga meddelanden genom att byta ut varje bokstav mot en version som flyttats ett visst antal steg framåt i alfabetet. En kodnyckel angav vad bokstaven 'A' skulle kodas som, och sedan kodade man alla bokstäver genom att skifta dem lika många steg framåt. Med kodnyckeln `C` skulle alltså alla `A` bli `C`, alla `B` blir `D`, och så vidare. Varje bokstav i meddelandet skiftas lika många steg. Mot slutet börjar alfabetet sedan om, och med nyckeln ovan skulle `X` bli `Z`, `Y` bli `A` och `Z` bli `B`.

Vi bryr oss här bara om bokstäverna A-Z, och gör inte skillnad på gemener och versaler. All text görs om till versaler.

Skriv en funktion `caesar_decoded(msg, key)` som tar ett meddelande och en nyckel, och returnerar ett meddelande där alla bokstäver har avkodats. Andra tecken (som inte är bokstäver) lämnas som de är.

Så här ska den fungera:

```
>>> caesar_decoded('YL DWWDFNHUDU L JUBQLQJHQ!', "A") # Om A kodades som A.
'YL DWWDFNHUDU L JUBQLQJHQ!'
>>> caesar_decoded('YL DWWDFNHUDU L JUBQLQJHQ!', "B") # Om A kodades som B.
'XK CVVCEMGTCT K ITAPKPIGP!'
>>> caesar_decoded("Yl dwwdfnhudu l jubqlqhjq", "B") # Gemener/versaler irrelevant.
'XK CVVCEMGTCT K ITAPKPGIP'
>>> caesar_decoded('YL DWWDFNHUDU L JUBQLQJHQ!', "D")
'VI ATTACKERAR I GRYNINGEN!'
```

Tips

- Funktionen `ord` returnerar heltal som motsvarar varje tecken, och `chr` returnerar tecken motsvarande heltal. `ord('A')` är `65`, och `chr(65)` är `'A'`.
- Strängmetoden `isalpha` returnerar `True` om dess argument är en bokstav. `"!".isalpha()` returnerar `False`.

make_circle_segment (1p)

Alex spelar ett välkänt datorspel och ska konstruera en cirkel av perfekt kvadratiska lådor, det hon vet är radien på cirkeln men hon vet inte hur hon ska bygga den. Givetvis vet Alex att en cirkel är symmetrisk, och eftersom världen är gjord av kvadratiska lådor finns det symmetri i både X och Y axeln, därför behöver hon bara ett "hörn" av en cirkel.

Skriv en funktion `make_circle_segment(r)`, som givet en radie `r`, returnerar en lista med listor i (en `r` × `r` matris), där varje element är en etta eller en nolla som säger om det ska vara ett block i den rutan eller inte. Segmentet som returneras av funktionen ska motsvara det nedre högra "hörnet" av cirkeln.

Exempel

```
>>> make_circle_segment(5)
[
    [1, 1, 1, 1, 1],
    [1, 1, 1, 1, 1],
    [1, 1, 1, 1, 1],
    [1, 1, 1, 1, 0],
    [1, 1, 1, 0, 0]
]
```

Det första elementet i den första inre listan har koordinaterna $(0, 0)$.

```
>>> make_circle_segment(10)
[
    [1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
    [1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
    [1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
    [1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
    [1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
    [1, 1, 1, 1, 1, 1, 1, 1, 1, 0],
    [1, 1, 1, 1, 1, 1, 1, 1, 0, 0],
    [1, 1, 1, 1, 1, 1, 1, 1, 0, 0],
    [1, 1, 1, 1, 1, 1, 0, 0, 0, 0],
    [1, 1, 1, 1, 1, 0, 0, 0, 0, 0]
]
```

Tips Det euklidiska avståndet r för en punkt (x, y) från origo kan beräknas med $r = x^2 + y^2$.

get_month_tot_ab_for_county (1p)

Filen `ftgstat_oppna_2019.csv` innehåller statistik om företag och föreningar från Bolagsverket för år 2019. Data är kommaseparerat med följande kolumner:

- `AR`: År
- `MANAD`: Månad
- `HANDELSE`: 1=Nyregistrerade, 2=Alla registrerade, 3=Avslutade
- `REGFAM`: 0=Regionfamilj saknas, 1=Storstadsregioner, 2=Större regioncentra, 3=Mindre regioncentra, 4=Små regioner- privat sysselsättning, 5=Små regioner-offentlig sysselsättning
- `REGFAMTEXT`: Regionfamilj
- `SATELAN`: Länskod
- `SATEKOMMUN`: Kommunkod
- `LANTEXT`: Län
- `KOMTEXT`: Kommun
- `AB`: Antal Aktiebolag
- `BAB`: Antal Bankaktiebolag
- `BF`: Antal Bostadsföreningar
- `BRF`: Antal Bostadsrättsföreningar
- `EK`: Antal Ekonomisk föreningar
- `E`: Antal Enskilda näringsidkare
- `SE`: Antal Europabolag
- `FL`: Antal Filialer till utländska företag
- `FAB`: Antal Försäkringsaktiebolag
- `HB`: Antal Handelsbolag
- `I`: Antal Ideella föreningar som bedriver näring
- `KB`: Antal Kommanditbolag
- `KHF`: Antal Kooperativa hyresrättsföreningar
- `MB`: Antal Medlemsbanker
- `SF`: Antal Sambruksföreningar
- `SB`: Antal Sparbanker
- `TSF`: Antal Trossamfund
- `BFL`: Antal Utländska bankers filialer
- `OFB`: Antal Ömsesidiga försäkringsbolag
- `LADDATUM`: Laddatum
- `ARMANAD`: År och Månad, ÅÅÅÅMM

I statistiken finns information om tre typer av s.k. "händelser" (kolumen `HANDELSE`); nyregistrerade (1), alla registrerade, d.v.s. totalt antal (2) och avslutade (3). Statistiken är på kommunnivå, d.v.s. för varje månad så finns siffror för alla tre typer av händelser för varje kommun. Letar vi efter statistiken för `"Linköping kommun"` från oktober 2019 får vi följande tre rader (endast fram till kolumnen `AB`)

```
2019,10,2,2,Större regioncentra,5,80,Östergötlands län,Linköping kommun,7641
2019,10,1,2,Större regioncentra,5,80,Östergötlands län,Linköping kommun,66
2019,10,3,2,Större regioncentra,5,80,Östergötlands län,Linköping kommun,19
```

Motsvarande rader kommer finnas för t.ex. alla andra kommuner i Östergötlands län.

Din uppgift är att skriva funktionen `get_month_tot_ab_for_county(month, county, filepath)` som returnerar hur många registrerade aktiebolag (kolumnen `AB`) det finns i ett län (kolumnen `LANTEXT`) en viss månad (kolumnen `MANAD`).

Eftersom det är det totala antalet vi är ute efter, är det endast raderna med händelsen “alla registrerade” (2) som är intressanta.

Exempel

```
>>> get_month_tot_ab_for_county(10, "Östergötlands län", "ftgstat_oppna_2019.csv")
20371
>>> get_month_tot_ab_for_county(10, "Stockholms län", "ftgstat_oppna_2019.csv")
211912
>>> get_month_tot_ab_for_county(10, "Skåne län", "ftgstat_oppna_2019.csv")
74265
```

tot_by_department (1p)

Till denna uppgift hör filen `leverantorsfakturor-umea-201908.csv` som innehåller information om Umeå kommuns inkomna leverantörsfakturor från augusti 2019. Informationen är hämtad från `https://oppnadata.se/`.

Filen innehåller semikolonseparerad data och första raden i filen innehåller följande kolumnrubriker:

- `ar` (år och månad för konteringen)
- `forvaltning` (förvaltning som fått fakturan)
- `leverantor` (den leverantör som fakturerat)
- `kod` (leverantörens organisationsnummer)
- `konto` (kontot fakturan konterats på)
- `kontotext` (beskrivning av kontot)
- `summa` (fakturabeloppet exkl. moms.)
- `verifikationsnummer` (verifikationsnummer)

Er uppgift är att skriva funktionen `tot_by_department(filepath)` som får in sökvägen till en fil enligt ovanstående specifikation. Sökvägen fås som en sträng. Funktionen ska skriva ut namn på förvaltningsenheterna följt av hur mycket varje enhet fakturerats under hela perioden som filen gäller.

Utskriften ska vara formatterad enligt nedan. Ordningen på förvaltningsenheterna spelar dock ingen roll. Exemplet nedan visar inte alla förvaltningar och siffrorna är för hela 2019, inte bara augusti.

```
Teknisk nämnd: 1637895704.09
Kulturnämnd: 51585938.88
För/Grundskolenämnd: 507051709.08
Äldrenämnd: 292573904.33
Miljö- & hälsoskydds: 6981533.06
.
.
.
```

Tips: Använd den inbyggda funktionen `round(number, digits)` som rundar av `number` till `digits` antal decimaler.

Del 2 (svårare uppgifter)

longest_common_substring (1p)

En delsträng är en obruten sekvens av tecken i en sträng. T.ex. strängen `"abc"` har delsträngarna `""`, `"a"`, `"b"`, `"c"`, `"ab"`, `"bc"` och `"abc"`.

Skriv en funktion `longest_common_substring(string1, string2)`, som tar in två strängar och returnerar den *längsta gemensamma delsträngen*. Både strängen `"python"` och strängen `"ton"` innehåller delsträngarna `""`, `"t"`, `"o"`, `"n"`, och `"on"`.

Anropet `longest_common_substring("python", "ton")` ska då returnera den längsta av dem: `"on"`. Om det finns fler än en längsta delsträng så kan du själv välja vilken av dem som returneras.

Exempel

```
>>> longest_common_substring("abc", "bcd")
'bc'

>>> longest_common_substring("aba", "abcba")  # 'ba' hade också varit ok
'ab'

>>> longest_common_substring("abc", "xyz")
''
```

OBS! Det finns inget krav på att din lösning är optimal.

exists_route (1p)

Denna uppgift handlar om problemlösning och indirekt om grafalgoritmer (något som du kommer att läsa om senare).

Vi ges en uppsättning flygrutter i ett dictionary på formatet

```
routes = { 'stockholm' : ['london', 'kastrup'],
           'london' : ['kastrup', 'stockholm'],
           'kastrup': ['new york'],
           'new york': ['kastrup', 'ulan bator'],
           'ulan bator' : [] }
```

I exemplet ovan ger `routes['stockholm']` listan av alla platser man kan flyga till direkt från `'stockholm'`. Vi ser också att det går att flyga från `'new york'` till `'ulan bator'`, men inte det omvända. Startpunkter är alltså nycklar i dictionaryt och destinationer är värden i listorna.

Skriv en funktion `exists_route(src, dst, routes)` som tar en startpunkt, en destination, och ett dictionary med rutter som argument. Funktionen ska returnera `True` om det finns ett sätt att ta sig från `src` till `dst`. Annars returneras `False`. Om ingen information om antingen `src` eller `dst` finns i dictionaryt returneras alltså `False`.

- En resa kan ha godtyckligt många mellanlandningar. Med exemplet ovan skulle `exists_route('stockholm', 'ulan bator', routes)` returnera `True` eftersom man kan flyga `'stockholm'` → `'kastrup'` → `'new york'` → `'ulan bator'`.
- Alla tillgängliga flygrutter antas finnas i det dictionary som funktionen får som argument. Om man inte kan flyga *från* en viss startpunkt, kommer dess värde vara en tom lista (t.ex. `'ulan bator'` ovan). Alla destinationer (alla värden i listorna) kommer finnas med som en startpunkt (en nyckel).

Tips

- För att ditt program inte ska fastna och leta i cykler som t.ex. `'stockholm'` → `'london'` → `'stockholm'` → ... kan det vara praktiskt att spara information om vilka flygplatser som redan "besökts" i en viss sökning.

find_seating (1p)

Boka biobiljetter online har nog många av er gjort. Denna uppgift går ut på att ta fram möjliga platser för ett sällskap av godtycklig storlek så att de kan *sitta tillsammans på samma rad*.

En biosalong har ett antal rader med sittplatser, men alla rader har inte nödvändigtvis lika många stolar. Vi representerar biosalongen som en lista där varje rad också representeras av en lista. De inre listor kan ha olika längd. Nedan ser ni två biosalonger (radbryt och mellanslag tillagda för att förtydgliga).

```
theatre1 = [    [0,  0,  0,  0,  0,  0],
              [0,  0,  0,  0,  0,  0,  0],
              [0,  0,  1,  1,  1,  0,  0],
            [0,  0,  0,  1,  1,  0,  0,  0],
            [1,  1,  0,  1,  1,  1,  1,  0],
            [0,  0,  0,  0,  0,  0,  0,  0]]

theatre2 = [  [0,  0,  0,  0,  0,  0],
              [0,  0,  1,  1,  0,  0],
            [0,  0,  0,  0,  0,  0,  0],
            [0,  0,  1,  1,  1,  0,  0],
              [0,  1,  1,  1,  1,  0],
              [0,  0,  1,  1,  1,  0],
            [0,  0,  0,  0,  0,  0,  0],
            [0,  0,  0,  0,  0,  0,  0]]
```

En tom plats representeras av värdet `0` och en upptagen plats representeras av värdet `1`.

Skriv funktionen `find_seating(theatre, num_people)` vars första argument, `theatre`, är en lista som representerar salongen enligt ovan. Andra argumentet är antalet personer i sällskapet. Funktionen ska returnera en lista med de förslag platser som sällskapet kan sitta på. Om det inte är möjligt för sällskapet att sitta tillsammans på samma rad returneras `[]`.

Varje sittplatsförslag är en tupel som innehåller tre heltal, t.ex. `(2, 2, 4)`. Det första heltalet är index för den rad som avses. Det andra heltalet är index för den första platsen på den raden, och det tredje heltalet är index för den sista platsen.

Tupeln `(2, 2, 4)` är ett förslag är att sällskapet ska sitta på raden med index `2`, på platserna med index `2`, till och med platsen med index `4` (sällskapets storlek är `3`).

Exempel Givet ovanstående värden på `theatre2` ska anropet `find_seating(theatre2, 7)` ska returnera listan `[(2, 0, 6), (6, 0, 6), (7, 0, 6)]`.

count_happy_tenants (1p)

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

Ovanstående nästlade lista representerar ett tvådimensionellt höghus. Talet `0` representerar en hyresgäst som är tystlåten och är bara glad om *alla* dess grannar är tystlåtna. Talet `1` representerar en hyresgäst som är högljudd och är bara glad om alla grannar är högljudda.

En hyresgästs grannar är de hyresgäster som bor ovanför eller bredvid, inte diagonalt. En hyresgäst kan alltså ha högst 4 grannar. En hyresgäst som bor i ett hörn har bara 2 grannar och en hyresgäst som bor på en kant har bara 3 grannar.

Skriv funktionen `count_happy_tenants(house)` vars argument `house` är en lista som representerar ett hus och dess hyresgäster enligt ovan. Funktionen ska returnera antalet glada hyresgäster, dvs summan av högljudda hyresgäster som bara har högljudda grannar, och tystlåtna hyresgäster som bara har tystlåtna grannar.

I huset högst upp finns det alltså totalt två glada hyresgäster: den högljudda längst ner till vänster och den tystlåtna på andra våningen, tredje lägenheten från vänster.

make_word_chain (1p)

Din uppgift är att skriva en funktion som försöker ordna ett antal ord så att varje efterföljande ord i sekvensen börjar på samma bokstav som det föregående ordet slutar på.

Orden ska användas exakt en gång och samma ord kommer inte förekomma mer än en gång bland input-orden.

För orden

- len
- drönare
- nej
- el
- jord

skulle en sekvens kunna vara *len, nej, jord, drönare, el.* En annan giltig sekvens är *el, len, nej, jord, drönare.*

Skriv funktionen `make_word_chain(words)` som får in en lista med ord och returnerar en lista med där orden kommer i en sekvens som uppfyller regeln. Om det inte går att bilda en giltig sekvens ska funktionen returnera en tom lista.

Tips: Det är väldigt svårt att lösa uppgiften med endast en funktion.

Exempel

- `make_word_chain(["till", "nit"])` returnerar `["nit", "till"]`
- `make_word_chain(["till", "nit", "bit"])` returnerar `[]`
- `make_word_chain(["till", "nit", "bit", "lin"])` returnerar `["bit", "till", "lin", "nit"]`

paint_bucket (1p)

Vi har en nästlad lista som representerar en svartvit bild. `0` är svart och `1` är vit. Vi har inga andra färger. Exempel på bilder:

```
picture1 = [[0, 0, 0, 0, 0, 0, 0],
            [0, 0, 1, 1, 1, 1, 1],
            [0, 0, 1, 0, 1, 0, 0],
            [1, 1, 1, 0, 1, 0, 0],
            [0, 0, 1, 1, 1, 0, 0]]

picture2 = [[0, 0, 1, 0, 1, 0, 0],
            [0, 0, 1, 1, 1, 0, 0],
            [0, 0, 1, 0, 0, 0, 0],
            [0, 0, 1, 0, 0, 0, 0],
            [0, 0, 1, 1, 0, 0, 0]]
```

Skriv funktionen `paint_bucket(picture, x, y, color)` ska användas för att hälla färgen `color` i koordinaten `x`, `y`. Koordinatsystemet är orienterat så att för bilderna ovan är det övre vänstra hörnet (0,0) och det nedre högra hörnet är (6,4).

Om vi häller vit färg (`1`) ändrar den färgen i rutan den börjar på och fortsätter sprida sig så länge den inte stöter på en vit ruta. Färg kan *inte* sprida sig diagonalt.

Funktionen ska returnera den den färdiga bilden.

Exempel Exemplena nedan gäller för ovanstående `picture1` och `picture2`.

- `paint_bucket(picture1, 0, 0, 1)` ger oss

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

- `paint_bucket(picture1, 0, 0, 0)` ger samma bild som den ursprungliga.
- `paint_bucket(picture2, 3, 0, 1)` ger oss

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

- `paint_bucket(picture2, 3, 0, 0)` ger oss samma bild som den ursprungliga.
- `paint_bucket(picture2, 3, 1, 0)` ger oss en helt svart bild.