TDDE44. DAT1. 2hp.

2024-05-30 kl. 8.00-14.00

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 1.1

- 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 1.1 i en fil med namnet `uppg11.py`.
- Filen skickas in via Studentklienten som '*Assignment #1*'.

Uppgift 1.1.1 - lös med lämplig loop

Skriv funktionen `sum_divisible(values, divisor)` som ska returnera summan av alla tal i `values` som är jämt delbara med `divisor`. Antag att alla element i `values` är heltal och att `divisor` är ett heltal större än 0. Den inbyggda funktionen `sum` får inte användas.

Exempel

```
>>> sum_divisible([2, 3, 4, 5, 6, 7, 8], 3) # (3 + 6)
9
>>> sum_divisible([5, -7, 2, -5, -6], 4)
0
>>> sum_divisible([-2, -9, 5, 5, 4, 5, -8, -1, 10, 2], 2)
6
>>> sum_divisible([4, -7, 2, -5, -8, -3, 2], 2)
0
```

Uppgift 1.1.2 - lös med lämplig loop

Skriv funktionen `deal_twentyone()` som returnerar en lista som innehåller slumpade heltal enligt följande: Tal mellan 1-11 (inklusive 1 och 11) ska läggas till en lista tills summan av talen är 21 eller högre, då listan returneras.

Du kan använda `random.randint(a, b)` från modulen `random` som returnerar ett slumpmässigt heltal mellan `a` och `b` (inklusive `a` och `b`).

Exempel

```
>>> deal_twentyone()
[9, 11, 2]
>>> deal_twentyone()
[7, 3, 3, 1, 1, 2, 7]
```

Uppgift 1.2 - lös med rekursion

- Spara lösningen till Uppgift 1.2 i en fil med namnet `uppg12.py`.
- Filen skickas in via Studentklienten som '*Assignment #2*'.

Skriv funktionen `tree_depth_rec(tree)` som tar ett träd representerat som nästlade listor. Beräkna trädets djup, dvs det maximala antalet nivåer av nästling som förekommer. Värdena på faktiska löv spelar ingen roll.

Exempel

```
>>> tree_depth_rec([1])
0
>>> tree_depth_rec([[2], 3])
1
>>> tree_depth_rec([['a'], 5, [['b'], [7.0]], 8])
2
>>> tree_depth_rec([[], [[]], [[[]]]])
3
```

Uppgift 1.3

- Spara lösningen till Uppgift 1.3 i en fil med namnet `uppg13.py`.
- Filen skickas in via Studentklienten som '*Assignment #3*'.

En databas med recept kan representeras av ett dictionary på följande sätt:

```
recept_db = {'kladdkaka': ['smör', 'socker', 'kakao', 'mjöl'],
             'typ bröd': ['mjöl', 'vatten'],
             'typ kaka': ['mjöl', 'socker', 'mjölk'],
             'sockerkaka': ['ägg', 'socker', 'smör', 'mjöl'],
             'pannkaka': ['ägg', 'socker', 'mjöl', 'mjölk'],
             'mördegskakaor': ['smör', 'socker', 'mjöl']}
```

Skriv en funktion `check_ingredients(database, ingredients)` som får en receptdatabas och en lista med ingredienser som argument och returnernar alla recept som kan bakas med dessa ingredienser (alla ingredienserna måste inte användas).

Exempel

Nedanstående är givet ovanstående värde på `recept_db`. **Obs! Ordningen på resultatet måste inte matcha exemplen nedan.**

```
>>> check_ingredients(recept_db, ['smör', 'socker', 'kakao', 'mjöl', 'vatten'])
['kladdkaka', 'typ bröd', 'mördegskakaor']
>>> check_ingredients(recept_db, ['smör', 'socker', 'mjöl', 'vatten'])
['typ bröd', 'mördegskakaor']
>>> check_ingredients(recept_db, ['öl', 'ost', 'kalles kaviar', 'nudlar'])
[]
>>> check_ingredients(recept_db, ['vatten', 'ost', 'kalles kaviar', 'mjöl'])
['typ bröd']
```

Uppgift 1.4

- Spara lösningen till Uppgift 1.4 i en fil med namnet `uppg14.py`.
- Filen skickas in via Studentklienten som '*Assignment #4*'.

I ett projekt har du fått i uppgift att implementera klassen `PictureData` som innehåller metadata för en bild. Informationen som ska lagras är filnamn, bredd och höjd. Nedan följer en beskrivning av klassen:

- När en instans skapas av klassen måste man ange parametrarna `filename` (bildfilens filnamn),`width` (bildens bredd) och `height` (bildens höjd). T.ex. `PictureData('strand.jpg', 1600, 900)` för bilden med filnamnet `'strand.jpg'`, bredden `1600` och höjden `900`.
- Klassen ska ha metoden `get_area()` som returnerar bildens area (totala antalet pixlar).
- Klassen ska ha metoden `copy()` som returnerar en ny instans av `PictureData` som har samma värden på sina instansvariabler som orginalet.

Implementera klassen `PictureData` enligt ovanstående specifikation.

Uppgift 1.5

- Spara lösningen till Uppgift 1.5 i en fil med namnet `uppg15.py`.
- Filen skickas in via Studentklienten som '*Assignment #5*'.

Denna uppgift använder filerna `partimandat2022.csv` och `bokstav_varde1.csv`

Skriv funktionen `csv_to_barchart(filename)` vars argument är sökvägen till en CSV-fil. Funktionen ska läsa data från CSV-filen och returnera en sträng som när den skrivs ut ritar upp ett liggande stapeldiagram av datat.

Du kan förutsätta att csv-filen kommer innehålla 2 kolumner där första kolumnen är namnen på staplarna och andra kolumnen är storleken på datan.

Din lösning ska klara av godtyckligt många rader i csv-filen och följa formatet i exemplet nedan (ett `#`-tecken representerar 2 hela enheter, d.v.s. 7 enheter blir `"###"`).

Dela upp din lösning i flera funktioner efter behov.

Exempel:

```
>>> chart = csv_to_barchart("partimandat2022.csv")

>>> print(chart)
 S: ################################################### (107 st)

SD: ################################### (73 st)

 M: ################################# (68 st)

 V: ############ (24 st)

 C: ############ (24 st)

KD: ######### (19 st)

MP: ######### (18 st)

 L: ######## (16 st)
```

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 2.1 (5p)

- Spara svaret till Uppgift 2.1 i en fil med namnet `uppg21.py`.
- Filen skickas in via Studentklienten som '*Assignment #6*'.

Alex spelar ett välkänt datorspel och ska konstruera en fylld cirkel av kubiska byggstenar, det hen vet är radien på cirkeln men hen vet inte hur hen ska bygga den. Givetvis vet Alex att en cirkel är symmetrisk, och eftersom spelvärlden består av kuber så finns det symmetri i både X och Y axeln, därför behöver hen bara en kvadrant (ett "hörn") av en cirkel.

Skriv en funktion `make_circle_segment(r)`, som givet en radie `r`, returnerar en $r \times r$ matris representerad som en lista med listor i, där varje element är en etta eller en nolla som säger om det ska vara ett block i den rutan eller inte. Matrisen som returneras av funktionen ska motsvara den nedre högra kvadranten av cirkeln. Se de bifogade bilderna `grid_circle5.png` till `grid_circle10.png` för illustrationer över hur ett rutnät kan approximera en cirkelkvadrant.

Tips 1: Skriv en funktion `print_mat` som skriver ut en matris representerad som en lista av listor på ett lättinspekterat sätt, t.ex. som nedan.

Tips 2: Kom ihåg att det euklidiska avståndet från origo för en punkt (x, y) kan beskrivas som hypotenusan i en rätvinklig triangel där kateterna har längden x och y.

Exempel

```
>>> make_circle_segment(5)
[[1, 1, 1, 1, 1], [1, 1, 1, 1, 1], [1, 1, 1, 1, 0], [1, 1, 1, 1, 0], [1, 1, 0, 0, 0]]
>>> print_mat(make_circle_segment(5))
[ [1, 1, 1, 1, 1],
  [1, 1, 1, 1, 1],
  [1, 1, 1, 1, 0],
  [1, 1, 1, 1, 0],
  [1, 1, 0, 0, 0] ]
>>> print_mat(make_circle_segment(8))
[ [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, 0],
  [1, 1, 1, 1, 1, 1, 0, 0],
  [1, 1, 1, 1, 1, 0, 0, 0],
  [1, 1, 1, 0, 0, 0, 0, 0] ]
```

Uppgift 2.2 (5p)

- Spara svaret till Uppgift 2.2 i en fil med namnet `uppg22.py`.
- Filen skickas in via Studentklienten som '*Assignment #7*'.

Vi representerar information om personer och deras barn med ett dictionary. Exempel (se bild på nästa sida för grafisk representation samma familjeträd):

```
family = {
  'ada': ['adam', 'arthur'],
  'boris': ['sven', 'selina', 'saga'],
  'adam': ['maria', 'magnus'],
  'sven': ['nathalie', 'nisse'],
  'maria': ['alva', 'alfred'],
  'david': ['alva', 'alfred'],
  'magnus': ['emma', 'erik'],
  'nathalie': ['emma', 'erik'],
  'alva': ['astrid']
}
```

I exemplet har t.ex. `'ada'` de två barnen `'adam'` och `'arthur'`. Varje nyckel är namnet på en person och varje värde är en lista som innehåller namnen på personens barn. Om en person inte har några barn, finns ingen nyckel för den personens namn. Den data vi har är dock inte komplett, så vi vet inte alltid namnen på båda föräldrarna till ett barn. I vårt universum får ett namn bara användas en gång i historien så det finns t.ex. bara en person som heter `'alfred'`.

Skriv funktionen `is_descendant(young_person, old_person, family_tree)` där `young_person` och `old_person` är namn och `family_tree` är ett dictionary enligt ovan. Funktionen ska returnera `True` om `young_person` är en ättling till `old_person` (godtyckligt antal led givet den information som finns). Om `young_person` inte är ättling till `old_person` returneras `False`.

Dela upp din lösning i fler funktioner efter behov.

Exempel

```
>>> is_descendant('ada', 'arthur', family)
False
>>> is_descendant('arthur', 'ada', family)
True
>>> is_descendant('maria', 'ada', family)
True
>>> is_descendant('maria', 'arthur', family)
False
>>> is_descendant('astrid', 'maria', family)
True
>>> is_descendant('astrid', 'david', family)
True
```

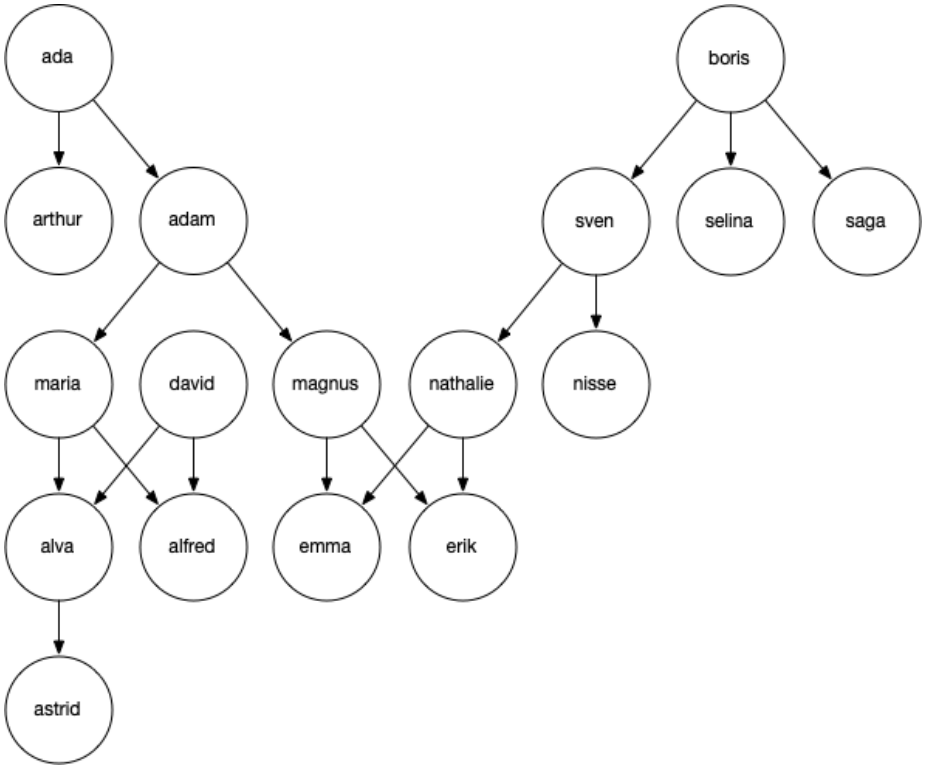

Figure 1: Familjeträd motsvarande exemplet i `is_descendant`-uppgiften.

Uppgift 2.3 (5p)

- Spara svaret till Uppgift 2.3 i en fil med namnet `uppg23.py`.
- Filen skickas in via Studentklienten som '*Assignment #8*'.

Python är bara ett av många programmeringsspråk i världen. Ett annat festligt språk är Lisp, som fram till 2019 användes i kursens föregångare. I Python skrivs matematiska uttryck med så kallad "infixnotation" där operatorn står *mellan* operanderna (som vi är vana vid från matematiken), t.ex. `(1 + 2)`.

I Lisp använder man istället så kallad "prefixnotation" där operatorn står *före* operanderna. `(1 + 2)` i prefixnotation blir alltså `(+ 1 2)`.

Här är ett exempel på ett större uttryck i först infixnotation och sedan prefixnotation:

```
# infixnotation
((1 + 2) / 3) + (4 * (5 - 6))
```

```
# prefixnotation (radbrytningar och indentering tillagda för
# tydlighetens skull)
(+ (/ (+ 1 2)
      3)
   (* 4
      (- 5 6)))
```

Er uppgift är att skriva en funktion `eval_prefix(expression)` som tar ett uttryck, representerat som en nästlad tupel som ovan (men med operatorerna som strängar), och returnerar värdet man får om man beräknar det.

Endast operatorerna `'+'`, `'-'`, `'*'`, och `'/'` kommer finnas, och varje operator har exakt 2 operander. **Obs! Notera att operatorerna är strängar.**

Exempel

```
>>> eval_prefix(('+', 1, 2))
3
>>> eval_prefix(('+', ('*', 3, 4), ('*', 5, 6)))
42
>>> eval_prefix(('+', ('/', ('+', 1, 2), 3), ('*', 4, ('-', 5, 6))))
-3.0
```