TDDE44. Exempeltenta. DAT1, 2hp.

2019-05-07

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

Skriva kod

Din kod för varje uppgift inklusive deluppgifter sparas i en fil. Exempel:

- Svaren för uppgift 1.1.1, 1.1.2 samt 1.1.3 sparas i en och samma fil.
- Svaren för uppgift 1.2.1, 1.2.2 samt 1.2.3 sparas i en och samma fil.

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

Inlämning av svar på uppgifter

TODO

Betyg på tentan

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

Rättning/bedömning av Del 1

För att får 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 studenten samlat ihop minst **2 poäng av 3 möjliga poäng** från uppgiftens deluppgifterna.
- Varje uppgift (t.ex. 1.1) består av 3 deluppgifter (t.ex. 1.1.1, 1.1.2, 1.1.3).
- En student kan få antingen 0; 0,5 eller 1 poäng per deluppgift.

Rättning/bedömning av Del 2

Del 2 rättas inte om inte Del 1 inte är godkänd.

Del 2 består av fyra frågor som är något mer omfattande och som kräver mer problemlösning än uppgifterna i Del 1. Fråga 2.1 och 2.2 kommer vara av samma svårighetsgrad. Fråga 2.3 och 2.4 kommer vara svårare än 2.1 och 2.2.

- Varje uppgift i del 2 kan få 0; 0,5 eller 1 poäng. **Det går att gå full poäng trots att någon mindre bugg finns om lösningen i helhet är bra**.
- Avdrag på 0,5 poäng kan ges för kod som bedöms vara mycket onödigt komplicerad, extremt svårläst, eller där konstruktioner används på fel sätt (till exempel använda en `while`-loop istället för `if`)

Frågor under tentan

TODO

Notation 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")`.

Exempel i uppgifter

Tänk på att eventuella exempel på anrop *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` behöver du inte oroa dig.

Del 1

Uppgift 1.1

Skicka in denna uppgift i Studentklienten som "Uppgift #1".

Uppgift 1.1.1

Skriv funktionen `sum_for(integer_list)` som ska returnera summan av alla heltal listan `integer_list` som är en lista av heltal. Funktionen ska inte använda den inbyggda funktionen `sum()` utan en `for`-loop som itererar över input-listan.

Anropet `sum_for([9, 3, 6, 3, 100, 2, 30])` ska alltså returnera `153`.

Uppgift 1.1.2

Skriv funktionen `sum_while(integer_list)` som ska returnera summan av alla heltal listan `integer_list` som är en lista av heltal. Funktionen ska inte använda den inbyggda funktionen `sum()` utan en `while`-loop som itererar över input-listan.

Anropet `sum_while([9, 3, 6, 3, 100, 2, 30])` ska alltså returnera `153`.

Uppgift 1.1.3

Skriv funktionen `create_int_list(start, stop, step)` som ska returnera en lista som innehåller en sekvens av heltal. Sekvensen ska börja med heltalet `start` och sluta med ett heltal som är mindre eller lika med `stop` (så nära `stop` som möjligt). Heltalen i sekensen ska öka i värde med värdet `step`.

Antag att alla argument till funktionen är positiva heltal, `step >= 1` och `stop >= start`.

Exempel

```
create_int_list(70, 3 80)
```

returnerar `[70, 73, 76, 79]` (`82` får inte vara med eftersom `82 > 80`)

Uppgift 1.2

Skicka in denna uppgift i Studentklienten som "Uppgift #2".

Uppgift 1.2.1

Skriv funktionen `keep_if_divisible_rec(lst, divider)` vars argument består av en lista och ett heltal. Vi gör antagandet att listan endast innehåller heltal och/eller flyttal. Funktionen ska **returnera** en ny lista med alla tal som är delbara med `divider` argumentet. Lös uppgiften rekursivt.

Exempel: `keep_if_divisible_rec([1,2,3,4], 2)` returnerar `[2,4]`

Uppgift 1.2.2

Skriv funktionen `even_range_product_rec(start, end)` vars argument är två heltal. Antag att `start <= end` och att det finns minst ett jämnt tal från `start` och `end`. Funktionen ska **returnera** produkten av alla jämna tal från `start` till och med `end`. Lös uppgiften rekursivt.

Exempel

`even_range_product(2,8)` returnerar `384` (= 2*4*6*8)

`even_range_product(2,2)` returnerar `2` (= 2*1)

Uppgift 1.2.3

Skriv funktionen `compare_value_after_n_years(cur_val1, rate1, cur_val2, rate2,n)` vars argument är positiva tal. `cur_val1` och `cur_val2` är två startvärden, `rate1` och `rate2` är årlig tillväxt för startvärde 1 respektive 2 (om `rate1 = 0.1` innebär det en tillväxt på `10%` per år för `cur_val1`). Funktionen ska **returnera** `1` eller `2` beroende på vilket värde som är störst efter `n` år av tillväxt.

Exempel: `compare_value_after_n_years(20, 0.1, 22, 0.01, 1)` returnerar `2` `compare_value_after_n_years(20, 0.1, 22, 0.01, 2)` returerar `1`

Uppgift 1.3

Skicka in denna uppgift i Studentklienten som "Uppgift #3".

Uppgift 1.3.1

Skriv funktionen `value_exists1(value, d)` där argumentet `value` är en `int` eller en `str`, och argumentet `d` är ett dictionary. Du kan anta att dictionaryts värden endast är av datatyperna `int` och `str`. Funktionen ska **returnera** `True` om argumentet `value` *finns* bland *dictionaryts värden*. Annars ska funktionen **returnera** `False`.

Exempel

```
value_exists1('hejsan', { 'a': 'bokstäver', 'b': 3 })
```

returnerar `False`.

Uppgift 1.3.2

Skriv funktionen `key_exists(key, d)` vars argument `d` är ett dictionary. Funktionen ska **returnera** `True` om argumentet `key` *finns* som *nyckel* i `d`. Annars ska funktionen **returnera** `False`.

Exempel

```
key_exists('blabla', {'blabla': 5})
```

returnerar `True`.

Uppgift 1.3.3

Vi har ett `dictionary` där nycklarna är namn på personer (strängar) och värdena är deras ålder. T.ex.

```
random_people = {"ada": 203, "guido": 63, "john": 79, "terry g": 78,
                 "eric": 76, "terry j": 77, "michael": 75}
```

Skriv funktionen `list_older(people, age)` där `people` är ett dictionary enligt ovan och `age` är ett heltal. Funktionen ska returnera en lista som innehåller alla namn på de personer som är äldre än `age`.

Exempel

```
list_older(random_people, 77)
```

kan returnera `["terry g", "john", "ada"]`. Eftersom ett dictionary inte har någon inombördes ordning mellan nycklarna är ordningen i listan godtycklig.

Uppgift 1.4

Skicka in denna uppgift i Studentklienten som "Uppgift #4".

Uppgift 1.4.1

Skriv funktionen `count_elements2(list_of_lists)` som får in en nästlad lista. De inre listorna innehåller inga listor. Funktionen ska returnera antalet element som inte är av datatypen `list` i både den yttre och de inre listorna.

Exempel

```
count_elements2([[1, 2], [3], 4, [1, 2, 3]])
```

returnerar `7`

Uppgift 1.4.2

Vi har ett dictionary var nycklar är namn på husdjur och vars värden också är dictionaries. Dessa dictionaries innehåller två nycklar vardera, `"age"` och `"species"`. Värdet som tillhör nyckeln `"age"` är husdjurets ålder och värdet som tillhör nycklen `"species"` är djurarten. T.ex.

```
my_pets = { "pluto": { "age": 4, "species": "cat" },
            "garfield": { "age": 7, "species": "dog" },
            "donald": { "age": 3, "species": "duck" },
            "felix": { "age": 12, "species": "cat"} }
```

Skriv funktionen `list_species(pets)` som får in en nästlad dictionary-struktur enligt ovan och returnerar en lista med alla djurarter som finns i dictionaryt. Inga dubletter får förekomma.

Exempel

```
list_species(my_pets)
```

kan returnera `["dog", "cat", "duck"]`. Eftersom det inte finns någon inbördes ordning mellan nycklar i ett dictionary är ordningen i den returnerade listan godtycklig.

Uppgift 1.4.3

Skriv funktionen `add_value(key, value, d)` där `d` är ett dictionary vars värden alltid ska vara listor, t.ex.

```
databas = { "färger": ["rosa", "grön"],
            "namn": ["Ada", "Bo", "Calle", "Disa"] }
```

Funktionen **ska inte returnera något**, utan ska ändra dictionaryt `d` (inte skapa något nytt dictionary) enligt följande:

- om argumentet `key` finns som nyckel i `d`, läggs argumentet `value` till den existerande listan som associerats med nyckeln `key`
- om argumentet `key` *inte* finns som nyckel i `d`, läggs argumentet `value` till i en ny lista som associeras med nyckeln `key`

Exempel

Givet ovanstående dictionary som refereras till med variabeln `databas`, görs följande två anrop:

```
# anrop 1
add_value("färger", "gul", databas)
# anrop 2
add_value("maträtter", "korv med bröd", databas)
```

Värdet i `databas` efter anrop 1:

```
{ "färger": ["rosa", "grön", "gul"],
  "namn": ["Ada", "Bo", "Calle", "Disa"] }
```

Värdet i `databas` efter anrop 1 följt av anrop 2:

```
{ "färger": ["rosa", "grön", "gul"],
  "namn": ["Ada", "Bo", "Calle", "Disa"],
  "maträtter": ["korv med bröd"] }
```

Uppgift 1.5

Skicka in denna uppgift i studentklienten som "Uppgift #5".

Uppgift 1.5

Till ett enkelt rymdspel behövs något sätt att hålla koll på var enstaka laserskott eller raketer befinner sig och hur de rör sig. Därför skriver vi klassen `Blast`. Varje skott har sin egna `x`- och `y`-position (som är tal). Tanken är att de i varje tidssteg ska ta ett steg i x-led (`dx`), och ett visst i y-led (`dy`). Eftersom skott kan ha olika riktning och hastighet, har varje skott sin egen `dx` och `dy`.

Laserskotten ska stödja följande metoder:

- Initialiseraren (`__init__`), som ska ta in x, y, dx och dy som argument.
- `get_position()` *returnerar* dess nuvarande position, som en tupel (x, y).
- `step()`. Detta uppdaterar skottets position ett steg (dess `x` blir `x + dx` och liknande i y-led).

Din uppgift är att

a) Skriva kod som definierar klassen `Blast` och

b) Skriva kod som skapar ett par `Blast`-instanser, skriver ut deras ursprungliga position (skriv ut tupeln med hjälp av print), tar ett par steg, och skriver ut deras nya positioner.

Del 2

Uppgift 2.1

Skicka in denna uppgift i studentklienten som "Uppgift #6".

Skriv funktionen `calculate(expression)` som tar in en sträng som innehåller en ett matematiskt uttryck som består av positiva heltal, plustecken och minustecken. Mellan de positiva heltalen och operatorerna finns det ett mellanslag. T.ex. `"10 + 7"` eller `"1 + 39 - 78 - 3 + 89"`.

Funktionen ska returnera det beräknade uttrycket som ett heltal.

Du får inte använda den inbyggda funktionen `eval()` för att lösa denna uppgift.

Uppgift 2.2

Skicka in denna uppgift i studentklienten som "Uppgift #7".

Ett palindrom är en sträng som ser likadan ut oavsett om man läser den framlänges eller baklänges. `KATT` är inte ett palindrom, för baklänges blir det `TTAK`. `APA` är däremot ett palindrom. Ett palindrom behöver inte vara ett ord, så `ADOLF I PARIS RAPAR SIRAP I FLODA` är ett palindrom (`ADOLF` blir `FLODA`, och så vidare). Här bryr vi oss bryr om alla tecken, så alla mellanslag, skiljetecken och så vidare måste också komma på rätt plats.

Skriv en funktion `is_palindrome(msg)` som tar en sträng och returnerar `True` om msg är ett palindrom, och `False` annars. Så här ska den fungera:

```
>>> is_palindrome("katt")
False
>>> is_palindrome("apa")
True
>>> is_palindrome("Apa")   # Gemener/versaler ska inte spela roll.
True
>>> is_palindrome("Adolf i Paris rapar sirap i Floda") # Alla strängar!
True
>>> is_palindrome("Apanapa")
True
>>> is_palindrome("Apan, apa") # Icke-bokstäver spelar roll!
False
```

Uppgift 2.3

Skicka in denna uppgift i studentklienten som "Uppgift #8".

Denna uppgift använder filerna `heltal_0-4a.txt` och `heltal_0-4b.txt`

I en fil finns data sparat så att varje rad innehåller ett heltal mellan 0 och 4. T.ex.

```
0
3
1
1
3
1
4
1
```

Skriv funktionen `barchart2(filename)` som läser in filen med sökvägen `filename` och returnerar en sträng som när den skrivs ut visar ett stapeldiagram ritat med `#`-tecken över datat. Den sista raden i utskriften ska innehålla

För exempelfilen skulle utskriften av den returnerade strängen se ut enligt följande:

```
 #
 #
 # #
## ##
01234
```

Du får dela upp din lösing på fler funktioner, men funktionen `barchart2(filename)` ska returnera den slutgiltiga strängen.

Uppgift 2.4

Skicka in denna uppgift i studentklienten som "Uppgift #9".

Denna uppgift handlar om problemlösning (och är skriven i en stil vanlig i programmeringstävlingar). För att göra invånarna i sin stad glada, har de styrande bestämt sig för att fylla igen hålen i den lokala Storgatan. Helst skulle man fylla alla hål, men budgeten är begränsad. Istället satsar man på att skapa en så lång sammanhängande vägsträcka helt utan hål som det bara går. Därför anlitas du.

Din uppgift är att skriva en procedur `longest_smooth_ride(road, budget)` som tar en *icke-tom* lista av tal som beskriver vägen (`road`), och ett heltal som anger hur många hål som kan repareras totalt (`budget`). Funktionen ska *returnera* längden av den längsta sammanhängande sträckan utan hål, efter att man utfört eventuella reparationer (på ett optimalt sätt). Det kan ibland finnas flera olika, men lika bra, sätt att reparera vägen, men längden kommer att vara densamma (annars vore de inte lika bra). En vägbeskrivning är en lista av talen 0 och 1 där 0 betyder att det är ett hål det vägsegmentet, och 1 betyder att det var ett helt vägsegment. Budgeten är ett heltal ≥ 0.

Så här ska din funktion fungera:

```
>>> longest_smooth_ride([0, 1, 1, 0, 1], 0)  # Ingen budget.
2
>>> longest_smooth_ride([0, 1, 1, 0, 1, 1], 1)  # Vilket av hålen fixas?
5
>>> longest_smooth_ride([0, 1, 1, 0, 1, 1, 0, 0], 2)
6
>>> longest_smooth_ride([0, 1, 1, 0, 1, 1], 999)  # Hela budgeten behöver inte användas.
6
```