TDDE44. Exempeltenta. DAT1, 2hp.

2023-13-37 kl. 24.00-28.00

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 två 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 + 4 poäng på Del 2.
- Betyg 5: Godkänd på Del 1 + 7 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.
- List comprehensions får *inte* användas för att lösa uppgifterna.
- Spara lösningarna till alla deluppgifter i Uppgift 1.1 i en fil med namnet `uppg11.py`.
- Filen skickas in via Studentklienten som '*Assignment #1*'.

1.1.1 - lös med `for`-loop

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 loop som itererar över input-listan.

Exempel

```
>>> sum_for([9, 3, 6, 3, 100, 2, 30])
153
```

1.1.2 - lös med `while`-loop

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 sedan ska efterföljande heltal öka med `step` och fortsätta så länge som talen är *mindre* än `stop`. Antag att alla argument till funktionen är heltal, och att `step >= 1`.

OBS! Funktionen `range()` får inte användas.

Exempel

```
>>> create_int_list(70, 82, 3)
[70, 73, 76, 79]
```

Uppgift 1.2 - lös med rekursion

- List comprehensions, `while` och `for` får *inte* användas för att lösa uppgiften.
- Spara lösningen i en fil med namnet `uppg12.py`.
- Filen skickas in via Studentklienten som '*Assignment #2*'.

Skriv funktionen `compare_value_after_n_years_rec(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_rec(20, 0.1, 22, 0.01, 1)
2
>>> compare_value_after_n_years_rec(20, 0.1, 22, 0.01, 2)
1
```

Uppgift 1.3

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

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 nyckeln `"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 mängd (använd datatypen `set`) med alla djurarter som finns i dictionaryt.

Exempel

```
>>> list_species(my_pets)
{'duck', 'cat', 'dog'} # Ordningen här är inte viktig
```

Uppgift 1.4

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

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 visst 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:

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

Exempel

```
>>> laser = Blast(x = 5, y = 3, dx = -1, dy = -1)
>>> laser.get_position()
(5, 3)
>>> laser.step()
>>> laser.step()
>>> laser.get_position()
(3, 1)
>>> goodtime = Blast(x = 0, y = 1, dx = 5, dy = 3)
>>> goodtime.get_position()
(0, 1)
>>> goodtime.step()
>>> goodtime.get_position()
(5, 4)
```

Uppgift 1.5

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

Skriv funktionen `calculate(expression)` som tar in en sträng som innehåller ett matematiskt uttryck som består av positiva heltal, plustecken och minustecken. Mellan de positiva heltalen och operatorerna finns det ett mellanslag.

Funktionen ska returnera det beräknade uttrycket som ett heltal.

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

Exempel

```
>>> calculate("10 + 7")
17
>>> calculate("1 + 39 - 78 - 3 + 89")
48
```

Del 2

- För att få betyg 4 krävs 4 poäng från denna del.
- För att få betyg 5 krävs 7 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*'.

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 2.2 (5p)

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

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 funktion `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 väg beskrivs av en lista som innehåller 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.

Exempel

```
>>> 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
```