729G46. Dugga. DUG1, 1hp.

2024-08-22 kl. 8.00-10.00

Se separat papper för information om inloggning, utloggning, inlämning av uppgifter och frågor under duggan.

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

Filer och kataloger

Istället för din vanliga hemkatalog får du en temporär hemkatalog under duggan. Du kommer fortfarande åt den med `~/`. Lägg kod du skriver direkt i denna katalog eller i en underkatalog du själv skapar. **Du kan inte spara din fil i `~/Desktop/given_files`**.

Skriva kod

Spara koden för alla uppgifter i varje del i en separat fil:

- Spara svaren till Del 1 filen `del1.py`.
- Spara svaren till Del 2 filen `del2.py`.
- Spara svaren till Del 3 filen `del3.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 duggan: Följande Python-moduler *tillåtna* att användas.

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

Godkänt på duggan

För att få godkänt på duggan måste alla delar vara godkända (Del 1, 2 och 3). Se info i anslutning till respektive del för hur många poäng som krävs för godkänt för den delen.

Rättning

Rättningen kommer avbrytas när en del inte uppnår det antal poäng som krävs för att den ska bli godkänd. Det vill säga om Del 1 är godkänd rättas Del 2. Om Del 1 och 2 är godkända rättas Del 3. Du kan få antingen 0, 1 eller 2 poäng per uppgift.

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")`.

- 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

- För att få godkänt på Del 1 krävs **5 av 8 poäng**.
- Spara dina lösningar på uppgifter från Del 1 i *en* fil med namnet `del1.py`.
- Filen skickas in via Studentklienten som '*Assignment #1*'.

create_greeting (2p)

Skriv funktionen `create_greeting(greeting, name)` som tar in en hälsningsfras (`greeting`) och ett namn (`name`). Båda argumenten är strängar. Funktionen ska returnera en hälsningsfras som skapats genom att använda argumenten (se nedan).

Exempel

```
>>> create_greeting('Hej', 'Ada')
'Hej Ada!'
>>> create_greeting('Tjena', 'Bertil')
'Tjena Bertil!'
```

get_winner_from_list (2p)

Skriv funktionen `get_winner_from_list(ranking_list)` vars argument `ranking_list` är en lista som innehåller namn (strängar). Namnen är från en löptävling och är ordnade efter de tävlandes målgångstider, från den snabbaste till den långsammaste. Antag att `ranking_list` innehåller minst ett element.

Funktionen ska returnera namnet på den snabbaste löparen.

Exempel

```
>>> get_winner_from_list(['ada', 'bertil', 'cecilia'])
'ada'
```

is_shorter_than (2p)

Skriv funktionen `is_shorter_than(values, length)` vars argument `values` är en lista och `length` är ett heltal som är större än 0. Funktionen ska returnera `True` om antalet värden i `values` mindre än `length`, annars returneras `False`.

Exempel

```
>>> is_shorter_than([1, 2], 2)
False
>>> is_shorter_than([1, 2], 3)
True
```

split_bill (2p)

Skriv funktionen `split_bill(amount, num_people)` som ska returnera hur mycket varje person i ett sällskap ska betala av notan om de delar lika. Argumentet `amount` är hur mycket middagen kostar och `num_people` är hur många personer som ingår i sällskapet.

Exempel

```
>>> split_bill(925, 4)
231.25
>>> split_bill(499, 2)
249.5
```

Del 2

- För att få godkänt på Del 2 krävs **4 av 8 poäng**.
- Spara dina lösningar på uppgifter från Del 2 i *en* fil med namnet `del2.py`.
- Det är inte tillåtet att använda *list comprehensions* för att lösa uppgifterna.
- Filen skickas in via Studentklienten som '*Assignment #2*'.

get_words_shorter_than_len (2p)

Uppgiften ska lösas med en `for`-loop

Skriv funktionen `get_words_shorter_than_len(word_list, word_length)` där `word_list` är en lista som innehåller strängar och `word_length` är ett heltal > 1. Funktionen ska returnera en lista som innehåller alla strängar från `word_list` som är *kortare* än `word_length`.

Exempel

```
>>> get_words_shorter_than_len(['hej', 'hopp', 'bil', 'ö', 'in'], 3)
['ö', 'in']
>>> get_words_shorter_than_len(['hej', 'hopp', 'bil', 'ö', 'in'], 2)
['ö']
```

sum_divisible (2p)

Uppgiften ska lösas med en `for`-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
```

slicer (2p)

Uppgiften ska lösas med en `while`-loop

Skriv funktionen `slicer(values, start, stop)` som går igenom listan `values` och skapar en ny lista som innehåller elementen från `values` med index `k`, där `start` $\leq$ `k` $<$ `stop`. Du kan utgå ifrån att både `start` och `stop` är större eller lika med 0, samt att `stop` är mindre än `len(values)`.

OBS! Du får *inte* använda slice-notation för att lösa uppgiften (t.ex. `values[1:3]`) men använd gärna slice-notationen för att testa att din funktion ger samma resultat. Funktionen `range()` får inte heller användas.

Exempel

```
>>> slicer(['a', 'b', 'c', 'd', 'e'], 1, 3)
['b', 'c'] # alla värden med index k, där 1 <= k < 3
>>> slicer(['a', 'b', 'c', 'd', 'e'], 0, 2)
['a', 'b'] # alla värden med index k, där 0 <= k < 2
>>> slicer(['a', 'b', 'c', 'd', 'e'], 4, 4)
[] # alla värden med index k, där 4 <= k < 4
```

deal_twentyone (2p)

Uppgiften ska lösas med en `while`-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]
```

Del 3

- För att få godkänt på Del 3 krävs **4 av 8 poäng**.
- Spara dina lösningar på uppgifter från Del 3 i *en* fil med namnet `del3.py`.
- Det är inte tillåtet att använda *list/dictionary comprehensions* för att lösa uppgifterna.
- Filen skickas in via Studentklienten som '*Assignment #3*'.

list_older (2p)

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

```
random_people = {'brian': 83, 'eric': 81, 'john': 84,
                 'michael': 81, 'terry g': 83, 'terry j': 82, }
```

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, 82)
['brian', 'john', 'terry g']
```

get_num_likes (2p)

Skriv funktionen `get_num_likes(people_colors, color)` som får in information om vilka färger olika personer gillar och ska returnera antalet personer som gillar färgen `color` som ett heltal.

Dictionaryt med informationen om personer och deras favoritfärger har namn på personer (strängar) som nycklar och en lista med personens favoritfärger som värden utan dubbletter (lista med strängar).

Exempel

```
>>> pc = { 'p1': ['red', 'green'],
           'p2': ['black', 'pink', 'blue'],
           'p3': ['blue', 'yellow', 'pink'],
           'p4': ['yellow', 'green'] }
>>> get_num_likes(pc, 'pink')
2
>>> get_num_likes(pc, 'cyan')
0
```

keep_if_divisible_rec (2p)

Uppgiften ska lösas med rekursion

Skriv funktionen `keep_if_divisible_rec(lst, divider)` vars argument består av en lista, `lst`, och ett heltal, `divider`. Listan `lst` innehåller endast heltal > 0. Antag att `divider` > 0. Funktionen ska returnera en ny lista med alla tal som är jämt delbara med `divider`.

Tips: Kom ihåg att modulo-operatorn `%` kan användas för att testa delbarhet.

Exempel

```
>>> keep_if_divisible_rec([1, 2, 3, 4], 2)
[2, 4]
>>> keep_if_divisible_rec([1, 2, 3, 4], 5)
[]
```

years_to_cutoff_rec (2p)

Uppgiften ska lösas med rekursion

Skriv funktionen `years_to_cutoff(start_val, rate, cutoff)` vars argument består av tre positiva tal. `start_val` är ett startvärde, `rate` är årlig tillväxt (rate = 0.1 innebär årlig tillväxt på 10%) och `cutoff` definierar tröskelvärdet. Funktionen ska returnera hur många år det tar för värdet att överstiga tröskelvärdet. Uppgiften skall lösas med rekursion.

Exempel

```
>>> years_to_cutoff(100, 0.1, 115)
2
>>> years_to_cutoff(100, 0.1, 200)
8
>>> years_to_cutoff(120, 0.1, 200)
6
>>> years_to_cutoff(100, 0.05, 200)
15
>>> years_to_cutoff(120, 0.05, 200)
11
```