729G46. Dugga. DUG1, 1hp.

HT2021

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; 0,5 eller 1 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 **2 av 3 poäng**.
- Spara dina lösningar på uppgifter från Del 1 i *en* fil med namnet `del1.py`.
- Filen skickas in via Studentklienten som '*Assignment #1*'.

add_values (1p)

Skriv funktionen `add_values(val1, val2)` vars båda argument, `val1` och `val2` är heltal. Funktionen ska returnera summan av `val1` och `val2`.

Exempel

```
>>> add_values(294, 12)
306
```

first_in_list (1p)

Skriv funktionen `first_in_list(values)` vars argument `values` är en lista. Funktionen ska returnera det första elementet i `values`. Listan `values` som funktionen får som argument kommer alltid innehålla minst ett element.

Exempel

```
>>> first_in_list(["hej", 123, 0.4])
'hej'
```

all_but_last (1p)

Skriv funktionen `all_but_last(values)` vars argument `values` är en lista. Funktionen ska returnera en lista som innehåller alla värden i `values` förrutom det sista värdet. Antag att `values` innehåller minst ett element.

Exempel

```
>>> all_but_last([1, 2, 3])
[1, 2]
```

Del 2

- För att få godkänt på Del 2 krävs **3 av 5 poäng**.
- Spara dina lösningar på uppgifter från Del 2 i *en* fil med namnet `del2.py`.
- Filen skickas in via Studentklienten som '*Assignment #2*'.

get_unique_random_integers (1p)

Uppgiften ska lösas med en `while`-loop

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-1` $\leqslant$ `stop-start`.

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

Exempel

```
>>> get_unique_random_integers(1, 100, 10)
[93, 23, 6, 26, 90, 29, 59, 12, 15, 86]  # tio slumpmässigt valda heltal utan
                                         # dubletter
```

keep_if_divisible (1p)

Uppgiften ska lösas med en `while`-loop

Skriv funktionen `keep_if_divisible(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`.

Exempel

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

remove_all_strings (1p)

Uppgiften ska lösas med en `for`-loop

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]`

sum_of_ints (1p)

Uppgiften ska lösas med en `for`-loop

Skriv funktionen `sum_of_ints(value_list)` som ska returnera summan av alla heltal (datatypen `int`) i listan `value_list`. Funktionen ska inte använda den inbyggda funktionen `sum()`.

Exempel

```
>>> sum_of_ints([10, "20", "hej", 100, 5.0, "30"])
110
>>> sum_of_ints([])
0
```

get_max_int (1p)

Uppgiften får lösas med valfri loopkonstruktion

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

```
>>> get_max_int([-9, 3, 6, "a", 3, 100, 2, 30])
100
>>> get_max_int(["a", -9, -8, -100])
-8
```

Del 3

- För att få godkänt på Del 3 krävs **4 av 6 poäng**.
- Spara dina lösningar på uppgifter från Del 3 i *en* fil med namnet `del3.py`.
- Filen skickas in via Studentklienten som '*Assignment #3*'.

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 }
>>> get_popular_words(wf)
['förlåt', 'träd' ]
```

value_exists (1p)

Skriv funktionen `value_exists(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_exists('a', { 'a': 'bokstäver', 'b': 3 })
False
>>> value_exists(3, { 'a': 'bokstäver', 'b': 3 })
True
>>> value_exists('a', {})
False
```

get_titles_with_tag (1p)

Antag att vi har lagrat en lista över böcker på följande sätt ett Pythonprogram:

```
b_list = [ {"title": "Learning Python", "tags": ["programming", "computer science"] },
           {"title": "Coders", "tags": ["programming", "technology"] },
           {"title": "Cryptonomicon", "tags": ["sci-fi", "historical"] },
           {"title": "Axiom's End", "tags": ["sci-fi", "aliens"] } ]
```

Vi har alltså en lista med böcker, där böckerna är dictionaries med två nycklar, `"title"` och `"tags"` där `"title"` är kopplat till en sträng med namnet på

boken, och `"tags"` är kopplat till en lista med strängar av taggar som boken fått tilldelad till sig.

Skriv funktionen `get_titles_with_tag(tag, books)` vars argument `tag` är en tagg som en sträng och `books` är en lista som innehåller böcker enligt ovanstående struktur. Funktionen ska returnera en lista med boktitlar som har blivit taggade med `tag`.

Exempel

```
# Nedanstående exempel använder b_list definierad ovan.
>>> get_titles_with_tag("programming", b_list)
['Learning Python', 'Coders']
>>> get_titles_with_tag("horror", b_list)
[]
>>> get_titles_with_tag("aliens", b_list)
["Axiom's End"]
```

count_pets (1p)

Skriv funktionen `count_pets(people)` som ska returnera det totala antalet djur som personerna i listan `people` har 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

```
# my_friends definierat enligt ovan
>>> count_pets(my_friends)
6
```

count_strings (1p)

Uppgiften ska lösas med rekursion

Skriv funktionen `count_strings(value_list)` som ska returnera antalet element av datatypen `str` som finns i listan `value_list`. Det returnerade värdet ska vara ett heltal.

Exempel

```
>>> count_strings(["30", True, "True", 3, 3.0, 6, "a", 100, 2, 30])
3
>>> count_strings([])
0
```

double_characters (1p)

Uppgiften ska lösas med rekursion

Skriv funktionen `double_characters(string_value)` som får in en sträng och returnerar en sträng där varje tecken i `string_value` upprepas en extra gång. `string_value` kommer innehålla minst ett tecken.

Exempel

```
>>> double_characters("hej på dig!")
'hheejj  ppåå  ddiigg!!'
>>> double_characters("")
''
```