729G46. Dugga. DUG1, 1hp.

2025-13-37 kl. 24.00-26.00

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), en *enkelsidig* A4 med handskrivna anteckningar

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 deluppgifter i varje del i en separat fil:

- Spara svaren till Del 1 i filen `del1.py`.
- Spara svaren till Del 2 i filen `del2.py`.
- Spara svaren till Del 3 i 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 och rättning

Duggan består av tre delar. För att få godkänt på duggan måste **alla delar** vara godkända.

- För att få **godkänt** på del 1 måste minst två uppgifter vara godkända, en som använder `for` och en som använder `while`.
- För att få **godkänt** på del 2 och del 3 måste minst en uppgift från respektive del vara godkänd.
- 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 feedback du fått så be om ett förtydligande istället för att chansa.

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 t.ex. får `1.99999999` som svar vid divisionen `4/2.0` behöver du inte oroa dig.

Kom ihåg att testköra din kod

Inkompletta eller delvis felaktiga lösningar *kan i undantagsfall* godkännas vid granskningen efter duggan, men kom ihåg att testköra din kod innan du skickar in! *Filer som kraschar vid körning ger normalt sett komplettering utan närmare granskning.* 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.

Starta Thonny

I tentasystemet startar du Thonny genom att öppna ett terminalfönster och skriva:

```
$ 729G46
```

Del 1 - Iteration

- För att bli godkänd på Del 1 måste **minst 2 uppgifter** vara godkända, minst en ska vara löst med `for` och minst en med `while`.
- Spara alla dina lösningar på uppgifter från Del 1 i *en* fil med namnet `del1.py`.
- Filen skickas in via Studentklienten som '*Assignment #1*'.

Uppgift 1.1 - lös med lämplig loop

Skriv funktionen `validate_wordle(guess, correct)` som tar in två ord på *exakt fem bokstäver* var. Båda orden kommer vara skrivna i VERSALER.

Funktionen ska gå igenom gissningen (argumentet `guess`) och returnera en lista med följande värden:

- `'green'` om tecknet finns med på motsvarande index i det korrekta ordet (argumentet `correct`)
- `'yellow'` om tecknet finns med någonstans i det korrekta ordet, men på ett annat index
- `'red'` om tecknet inte finns med alls.

Obs! Markeringen i denna uppgift är förenklad jämfört med riktiga Wordle. Här markeras varje bokstav *alltid* enligt ovanstående regler oavsett om bokstaven förekommit/markerats tidigare eller inte (se sista exemplet där `'P'` bara förekommer en gång i `'POKER'` men markeras som `'yellow'` båda gångerna det förekommer i `'HAPPY'`).

Exempel

```
>>> validate_wordle('HEART', 'YOUNG')
['red', 'red', 'red', 'red', 'red']
>>> validate_wordle('GRUNT', 'YOUNG')
['yellow', 'red', 'green', 'green', 'red']
>>> validate_wordle('YOUNG', 'YOUNG')
['green', 'green', 'green', 'green', 'green']
>>> validate_wordle('HAPPY', 'POKER')
['red', 'red', 'yellow', 'yellow', 'red']
```

Uppgift 1.2 - lös med lämplig loop

Skriv funktionen `increment_ints(values, inc)` där `values` är en lista av strängar och tal och `inc` är ett tal. Funktionen ska returnera en lista motsvarande `values` men med alla heltal ökade med `inc`.

Exempel

```
>>> increment_ints([], 1)
[]
>>> increment_ints([1, 2, 3], 0)
[1, 2, 3]
>>> increment_ints([1, 2, 3], 1)
[2, 3, 4]
>>> increment_ints([1, 2, 3], 5)
[6, 7, 8]
>>> increment_ints(['a', 2, 'hej', 3.0, 6], 1)
['a', 3, 'hej', 3.0, 7]
>>> increment_ints(['a', 2, 'hej', 3.0, 6], 5)
['a', 7, 'hej', 3.0, 11]
>>> increment_ints(['a', '2', 'hej', 3.0], 5)
['a', '2', 'hej', 3.0]
```

Uppgift 1.3 - lös med lämplig 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 att `number_of_ints-1` $\leq$ `stop-start`.

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

Exempel

```
>>> get_unique_random_integers(10, 20, 1)
[19]                                    # ett slumpmässigt valt heltal
>>> get_unique_random_integers(1, 100, 10)
[84, 17, 76, 23, 80, 3, 70, 29, 13, 66]  # tio slumpmässigt valda heltal utan
                                        # dubletter
```

Uppgift 1.4 - lös med lämplig loop

Skriv funktionen `count_repetitions(characters)` som tar in en sträng. Funktionen skall returnera antalet gånger som tecken upprepas *i direkt följd*. Strängen `'aba'` har inga upprepningar eftersom inga tecken förekommer två gånger i direkt följd. Strängen `'abb'` har 1 upprepning eftersom `b` upprepas en gång. Strängen `'aaa'` har 2 upprepningar eftersom `a` först upprepas en gång och sedan upprepas en gång till. Strängen `'aabb'` har 2 upprepningar eftersom först upprepas `a` en gång och sedan upprepas `b` en gång.

Tips: När du itererar över strängen behöver du bara hålla koll på vad det senaste värdet var för att identifiera en upprepning.

Exempel

```
>>> count_repetitions('aaaaa')
4
>>> count_repetitions('aabaa')
2
>>> count_repetitions('aaabb')
3
>>> count_repetitions('abababa')
0
>>> count_repetitions('abcde')
0
>>> count_repetitions('aabbccddee')
5
```

Del 2 - Rekursion

- För att bli godkänd på Del 2 måste **minst 1 uppgift** vara godkänd.
- Spara alla dina lösningar på uppgifter från Del 2 i *en* fil med namnet `del2.py`.
- Filen skickas in via Studentklienten som '*Assignment #2*'.

Uppgift 2.1 - lös 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
```

Uppgift 2.2 - lös med rekursion

Skriv funktionen `double_characters_rec(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.

Uppgiften skall lösas med rekursion.

Exempel

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

Del 3 - Dictionaries och nästling

- För att bli godkänd på Del 3 måste **minst 1 uppgift** vara godkänd.
- Spara alla dina lösningar på uppgifter från Del 3 i *en* fil med namnet `del3.py`.
- Filen skickas in via Studentklienten som '*Assignment #3*'.

Uppgift 3.1

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

```
b_list = { 'Learning Python': {
                'author': 'Mark Lutz',
                'tags': ['programming', 'computer science']
           },
           'Weapons of Math Destruction': {
                'author': 'Cathy O\'Neil',
                'tags': ['computer science', 'society']
           },
           'Cryptonomicon': {
                'author': 'Neal Stephenson',
                'tags': ['sci-fi', 'historical']
           },
           'Axiom\'s End': {
                'author': 'Lindsay Ellis',
                'tags': ['sci-fi', 'aliens']
           }
        }
```

Vi har alltså en dict med böcker, där varje nyckel är en boktitel. Till varje boktitel hör en dict med två nycklar, `'author'` och `'tags'` där `'author'` är kopplat till en sträng med namnet på bokens författare, och `'tags'` är kopplat till en lista med strängar av taggar som boken blivit tilldelad.

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']
```

Uppgift 3.2

Skriv funktionen `has_pets(people, number)` som tar en lista av dictionaries enligt nedan och ett tal. 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.

Funktionen ska returnera en lista med namnen på de personer i listan `people` som har minst `number` husdjur

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
>>> has_pets(my_friends, 0)
['Ada', 'Bertil', 'Celeste']
>>> has_pets(my_friends, 1)
['Ada', 'Celeste']
>>> has_pets(my_friends, 2)
['Ada', 'Celeste']
>>> has_pets(my_friends, 3)
['Celeste']
```