729G46. Dugga. DUG1, 1hp.

2024-10-28 kl. 14.00-16.00

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 uppgift i en separat fil:

- Spara svaren till Uppgift 1 i filen `uppg1.py`.
- Spara svaren till Uppgift 2 i filen `uppg2.py`.
- Spara svaren till Uppgift 3 i filen `uppg3.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 uppgifter. För att få godkänt på duggan måste **alla uppgifter** vara godkända.

- För att få **godkänt** på en uppgift med deluppgifter (t.ex. uppgift 1) så måste samtliga deluppgifter vara godkända om inte annat anges.
- 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 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 0 poäng 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.

Starta Thonny

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

```
$ 729G46
```

Uppgift 1 - Loopar

- För att få godkänt på denna uppgift skall **båda** deluppgifterna lösas.
- En av uppgifterna skall lösas med en `while`-loop och en skall lösas med `for`-loop.
- Comprehensions och generatorer får *inte* användas för att lösa uppgifterna.
- Spara lösningarna till båda deluppgifterna i Uppgift 1 i en fil med namnet `uppg1.py`.
- Filen skickas in via Studentklienten som '*Assignment #1*'.

Uppgift 1.1 - lös med lämplig loop

Skriv funktionen `create_mult_table(int_val)` vars argument `int_val` är en `int`. Funktionen ska returnera multiplikationstabellen för `int_val`.

Multiplikationstabellen returneras som en lista med 10 strängar. Varje sträng är alltså på formatet '*faktor mellan 1-10**`int_val`=*produkt*'. Första faktorn i multiplikationstabellen är alltid 1 och den sista är alltid 10.

Lös uppgiften med en lämplig loop.

Exempel

```
>>> create_mult_table(5)
['1*5=5', '2*5=10', '3*5=15', '4*5=20', '5*5=25', '6*5=30',
 '7*5=35', '8*5=40', '9*5=45', '10*5=50']
>>> create_mult_table(9)
['1*9=9', '2*9=18', '3*9=27', '4*9=36', '5*9=45', '6*9=54',
 '7*9=63', '8*9=72', '9*9=81', '10*9=90']
>>> create_mult_table(2)
['1*2=2', '2*2=4', '3*2=6', '4*2=8', '5*2=10', '6*2=12',
 '7*2=14', '8*2=16', '9*2=18', '10*2=20']
>>> create_mult_table(6)
['1*6=6', '2*6=12', '3*6=18', '4*6=24', '5*6=30', '6*6=36',
 '7*6=42', '8*6=48', '9*6=54', '10*6=60']
```

Uppgift 1.2 - lös med lämplig loop

Skriv funktionen `get_investment_time(start, goal, interest)` som ska returnera antalet hela år det tar för startkapitalet `start` att nå eller överstiga värdet `goal` givet en fast årsränta `interest`.

Om årsräntan är 10 % kommer investeringen att ha ökat med 10 % efter första året. Efter andra året har totalen ökat med 10 % av föregående års värde inklusive räntan för det året, osv. T.ex. med en initial investering av 100 och en ränta på 10 % så tar det 8 år för investeringen nå ett värde av 200 (se exempel nedan).

Uppgiften ska inte lösas algebraiskt utan med hjälp av en lämplig loop. Antag att `goal` > `start` och att alla värden är positiva.

0bs: Notera att räntan är representerad som en kvot, dvs räntan 10 % representeras som flyttalet `0.10`, räntan 5 % som `0.05`, osv. Alltså kan värdet efter 1 år beräknas som `start + (interest * start)`.

Exempel

```
>>> get_investment_time(100, 120, 0.10)
2
>>> get_investment_time(100, 150, 0.10)
5
>>> get_investment_time(100, 180, 0.10)
7
>>> get_investment_time(100, 200, 0.10)
8
>>> get_investment_time(100, 120, 0.05)
4
>>> get_investment_time(100, 150, 0.05)
9
>>> get_investment_time(100, 180, 0.05)
13
>>> get_investment_time(100, 200, 0.05)
15
```

Uppgift 2 - Dictionaries och nästling

- För att få godkänt på denna uppgift skall **båda** deluppgifterna lösas.
- Spara lösningarna till båda deluppgifterna i Uppgift 2 i en fil med namnet `uppg2.py`.
- Filen skickas in via Studentklienten som '*Assignment #2*'.

Uppgift 2.1 - Dictionary

Skriv funktionen `find_students_with_passing_grades(students)` som får in ett dictionary `students` där nycklarna är namn (strängar) och värdena är strängar `'U'`/`'G'`/`'VG'` som står för betyget studenten med namnet fått. Funktionen ska returnera en lista med namnen på de personer som har fått `'G'`/`'VG'`. Du kan anta att `students` innehåller minst en person (med tillhörande betyg).

Exempel

```
>>> find_students_with_passing_grades({'ada': 'G',
                                       'boris': 'G',
                                       'cissi': 'VG',
                                       'doris': 'U'})
['ada', 'boris', 'cissi']
>>> find_students_with_passing_grades({'doris': 'U'})
[]
```

Uppgift 2.2 - Nästlad data

Skriv funktionen `get_pairs(values)` vars argument `values` innehåller godtyckligt många tupler som innehåller exakt två heltal. Funktionen ska returnera en lista som endast innehåller de tupler där båda elementen är lika.

Exempel

```
>>> get_pairs([(1, 2), (3, 3), (29, 23), (5, 5)])
[(3,3), (5, 5)]
>>> get_pairs([(5, 2), (3, 4), (29, 23), (98, 7)])
[]
>>> get_pairs([(3, 2), (2, 2), (2, 5), (3, 3), (1, 4), (2, 2)])
[(2, 2), (3, 3), (2, 2)]
```

Uppgift 3 - Rekursion

- Spara lösningen till Uppgift 3 i en fil med namnet `uppg3.py`.
- Filen skickas in via Studentklienten som '*Assignment #3*'.
- *Uppgiften ska lösas med rekursion*

Skriv funktionen `has_word_longer_than(words, length)` som returnerar `True` om det finns ett ord i listan `words` som är längre än `length` (ett heltal) antal tecken, annars returneras `False`. Antag att `words` endast innehåller strängar.

Obs! Uppgiften skall lösas med rekursion.

Exempel

```
>>> has_word_longer_than([], 3)
False
>>> has_word_longer_than(['hej', 'nej'], 3)
False
>>> has_word_longer_than(['mjöl', 'mjölk', 'ägg', 'socker', 'salt'], 3)
True
```