Programmeringsteknik I: Tentamen

2020-10-23 08:00 - 13:00 + 30 minuter för att lämna in din lösning i Studium.

Välkommen till tentamen för kursen Programmeringsteknik I. Denna tentamen består av två delar, A och B. Den första delen innehåller ett antal problem som består av att lösa mindre problem, eller svara på frågor om kod. Den andra delen består av ett aningen större program, där du behöver komplettera dom delar som är utelämnade, samtidigt som du ser till att din lösning fungerar väl med den givna koden.

Lösningar från tidigare frågor kan användas i efterföljande frågor, om inget annat nämns, även om dom inte har lösts. Det är tillåtet (och ibland rekommenderat) att introducera nya metoder och funktioner. Instruktionen “Skriv en funktion som” betyder alltså inte att ditt svar måste struktureras utan extra hjälpfunktioner.

Alla frågor behandlar programmeringsspråket Python 3, och all kod ska skrivas i det språket. Extra hänsyn ska ges vad gäller läsbarhet (din kod skall vara väl strukturerad och korrekt indenterad). Variabel, funktion, metod och klassnamn skall vara beskrivande, men kan ofta hållas ganska korta. Notera att ditt betyg kan påverkas negativt av, bland annat:

- Onödiga variabler,
- Dålig läsbarhet,,
- Repetition av identiska stycken kod
- Misslyckande att använda given eller tidigare skriven kod på ett bra sätt
- Väldigt ineffektiv kod (t.ex. med många onödiga funktionsanrop)

Om du inte kommer ihåg exakt vad en funktion heter eller hur en del av Pythons syntax ser ut, kan du påpeka detta i en kommentar, och beskriva vilka antaganden du gör. Vi bedömer både hur du löser problem och hanterar programmeringsspråket Python.

Regler:
Du får söka på Google, StackOverflow, Python documentation och liknande källor på internet. Om du baserar din lösning på någon sådan resurs, ge en länk/referens till den.
Du får utöver detta köra din kod på din dator, och inspektera utskrifter (och använda debuggern, extra test-fall osv).
Du får under inga omständigheter kommunicera eller arbeta tillsammans med någon annan person (student eller någon annan) under tentan. Om du har frågor, kontakta huvudläraren på kursen, Johan Öfverstedt (johan.ofverstedt@it.uu.se). Fusk och plagiat kommer inte att tolereras och kommer resultera i diciplinära åtgärder, som kan innefatta avstängning från alla studier på Uppsala Universitet.

Betygsättning
För att bli godkänd på tentamen (betyg 3) är det nödvändigt att A-delen är mestadels rätt. Detta betyder inte att alla uppgifter behöver vara helt rätt gjorda, men du måste demonstrera att du uppfyller kursens mål.
För betyg 4, är det också nödvändigt att klara av hälften av uppgifterna på B-delen. Du behöver lösa samtliga uppgifter på B-delen mestadels korrekt för att få betyg 5.

Kodkvaliteten kommer också tas i beaktande för de högre betygen. Del B kommer inte att granskas om inte del A är godkänd.

Din lösning innehållandes alla svar ska skickas in på Studium i en enda .txt eller .py (om du gör koden helt körbar) fil till Studium.
Märk varje fråga med #A1, #A2, osv, så att sökning lätt kan göras i dokumentet.
Se till att skiva din anonyma tentakod överst i filen.

Lycka till!

Del A

A1. Beskriv i ord (färre än 100) vad följande kod gör, och lista alla

- moduler,
- funktionsdefinitioner,
- metoddefinitioner,
- funktionsanrop,
- metodanrop,
- klasser,
- variabler,
- attribut,
- konstruktorer,
- parametrar (inte argument).

```
import random

def fib(n):
    if n == 0:
         return 0
    if n == 1:
         return 1
    f0 = 0
    f1 = 1
    for _ in range(2, n):
        fi = f0 + f1
        f0 = f1
        f1 = fi
    return f1


class Dice:
    def __init__(self, nsides):
        self.nsides = nsides
        self.value = 0
```

```
    def roll(self):
        self.value = random.randint(1, self.nsides)

d = Dice(6)
d.roll()
print(fib(d.value))
```

A2. Skriv en funktion `alternating` som tar en sträng som parameter och returnerar en ny sträng, där varannat tecken är versal och gemen om vartannat.

Example:

```
print(alternating('This is all A part of the test.'))
# Should give the print-out: ThIs iS AlL A PaRt oF ThE TeSt.
```

A3. Skriv en funktion predecessors(s) som tar en sträng som parameter, delar upp den i ord (med metoden `split`), omvandlar varje ord till gemener (med metoden `lower`), och returnerar ett lexikon (dictionary) som innehåller, för varje ord, dom ord som förekommer ordet i texten.

Hint: Du behöver hålla reda på föregående ord när du loopar igenom orden i strängen, och föregående ord för det första ordet kan antas vara en tom sträng ".

Example:

```
print(predecessors('Hello world the world is the best world in the
world'))
#{'hello': [''], 'world': ['hello', 'the', 'best', 'the'], 'the':
['world', 'is', 'in'], 'is': ['world'], 'best': ['the'], 'in':
['world']}
```

A4. Skriv en funktion `central_difference(lst)` som tar en lista av tal (flyttal eller heltal) och för varje index i listan, beräknar en centraldifferens approximation g(x) = (f(x+1)-f(x-1)) / 2. Det första och sista elementet i den ursprungliga listan kan ignoreras; för en lista av 7 element som argument ska en lista med 5 element returneras. Den ursprungliga listan får inte förändras inuti funktionen.

Example:

```
print('central_difference([-5, 7, 4, 9, 10, 11, 0]):',
central_difference([-5, 7, 4, 9, 10, 11, 0]))
# should give the output:
central_difference([-5, 7, 4, 9, 10, 11, 0]): [4.5, 1.0, 3.0, 1.0,
-5.0]
```

A5. Skriv en funktion `random_shuffle(lst)` som tar en lista som argument och returnerar en ny lista som innehåller alla element som fanns i den ursprungliga listan, men i

en slumpmässigt utvald ordning (helst så att varje möjlig permutation är lika sannolik).

Du får använda funktioner från `random` modulen, eller `numpy.random`, men inte `numpy.random.random_shuffle` funktionen (eller liknande färdiggjort alternativ).
För att lösningen ska anses korrekt, måste den kunna returnera olika resultat när den anropas för samma lista (och uppfylla testerna som ska skrivas som en del av nästa fråga).

Exempel:

```
print(random_shuffle(['Hej', 5, [1, 2, 3], 'Python is great!']))
```
kan ge en utmatning som (men givetvis kan ge andra result då den är slumpmässig):
```
[5, [1, 2, 3], 'Python is great!', 'Hej']
```

A6. Skriv en funktion `test_random_shuffle(lst)` som tar en lista som parameter och ämnar att testa korrekthet hos den föregående funktionen `random_shuffle`.Testet måste verifiera att:

1) `random_shuffle` inte modifierar listan som ges som argument till funktionen (möjligtvis genom att göra en kopia som kan jämföras med argumentlistan efter funktionsanropet),
2) alla element i den ursprunliga listan har bevarats, men möjligtvis befinner dom sig på en annan plats i listan,
3) att den returnerade listan har samma längd som den ursprunliga listan.

A7. Skriv en funktion `draw_text_circle(r, n)` som för ett rutnät med 2*n+1 rutor (positioner på index -n to n) i varje dimension, ritar ut en cirkel till standard output (använd en nästlad for-loop med print-satser, med `end=''` för att undvika radbrytningar efter varje skriven symbol). Magnituden för en given position givs av formeln `sqrt(x*x + y*y)` och vi definerar att vara på cirkeln som att ha en position med en magnitud mellan r - 0.5 och r + 0.5.

Följande exempel skrivs ut när funktionen anropas som `draw_text_circle(7, 9)`

```
.................
.................
.......XXXXX.......
.....XX.....XX.....
....X.........X....
...X...........X...
...X...........X...
..X.............X..
..X.............X..
..X.............X..
..X.............X..
..X.............X..
...X...........X...
...X...........X...
....X.........X....
.....XX.....XX.....
.......XXXXX.......
.................
.................
```

A8. Vi ska nu skapa en klass som representerar en standard (digital) klocka som räknar från 00:00:00 till 23:59:59. Skriv klassdefinitionen, med namnet Clock, lägg till en konstruktor som tar emot en initialtid i form av tre parametrar (hour, minute, second) och lagrar dessa som attribut. Lägg sedan till en __str__ metod som returnerar en sträng med formatet 07:23:42 (inklusive ledande nollor).

Example:

```
# The following code:
c = Clock(7, 11, 13)
print(c)
# should give the following output:
07:11:13
```

A9. Skriv en metod `tick` i klassen Clock som tar en parameter n (med standardvärde 1) som avgör hur många sekunder klockan ska ökas med. Varje gång sekundräknaren visar 60 ska den vridas tillbaka till 0, samtidigt som minuträknaren ökas med 1. På samma sätt ska

timräknaren ökas varje gång minuträknaren blir 60, som också vrids tillbaka till 0. När timräknaren blir 24 ska den vridas tillbaka till 0.

Example:

```
# The following code:
c = Clock()
c.tick(7*60*60)
print(c)
c.tick(72)
print(c)
c.tick(17*60*60)
print(c)
# should give the following output:
07:00:00
07:01:12
00:01:12
```

Part B

Alla problem i den följande problemsekvensen handlar om att skriva en labyrintlösare.

Du får given följande kod:

```
class Maze:
    def __init__(self, n):
        self.a = [['X' for _ in range(n)] for _ in range(n)]

    def create_path_between(self, x1, y1, x2, y2):
        xc = x1
        yc = y1
        self.a[yc][xc] = ' '
        while xc != x2 or yc != y2:
            if x2 > xc:
                xc += 1
            elif x2 < xc:
                xc -= 1
            if y2 > yc:
                yc += 1
            elif y2 < yc:
                yc -= 1
            self.a[yc][xc] = ' '
```

```
    def create_exit(self, x, y):
        self.a[y][x] = 'E'

    def get_neighbour(self, x, y):
        if x < 0 or y < 0 or x >= len(self.a) or y >= len(self.a):
            return (x, y, 'X')
        else:
            return (x, y, self.a[y][x])

    def get_neighbours(self, x, y): # *** B1: get_neighbours
        pass # replace pass with your code

    def __str__(self):
      s = ''
      s += 'X' * (len(self.a)+2)
      s += '\n'
      for y in range(len(self.a)):
          s += 'X'
          for x in range(len(self.a)):
              s += self.a[y][x]
          s += 'X\n'
      s += 'X' * (len(self.a)+2)
      return s

mz = Maze(10)
mz.create_path_between(0, 0, 9, 9)
mz.create_path_between(1, 1, 7, 3)
mz.create_path_between(7, 3, 5, 0)
mz.create_path_between(7, 3, 8, 5)
mz.create_path_between(8, 5, 7, 6)

mz.create_path_between(0, 0, 0, 4)
mz.create_path_between(0, 4, 4, 7)
mz.create_path_between(4, 7, 0, 9)
mz.create_path_between(0, 9, 0, 5)

mz.create_exit(9, 9)

print(mz)

mz_solver = RandomMazeSolver(0, 0, mz) # *** B2: RandomMazeSolver

print(mz)
while mz_solver.step() == False: # *** B3: step
    pass

pth = mz_solver.path
```

```
shrt_pth = shorten_path(pth) # *** B4: shorten_path

print('Long path: ', pth, sep='')
print('Short path: ', shrt_pth, sep='')
print(f'Length of long path: {len(pth)}.')
print(f'Length of short path: {len(shrt_pth)}.')
```

När den körs ska följande utskrift ges (upp till slumpmässighet i den väg som väljs):

```
XXXXXXXXXXXX
X XXXX XXXXX
X  XXX XXXXX
X X XXX XXXX
X XX  XXX
X XXX XXX XX
X  XXX XX XX
X X XXX  XXX
X XX  XX XXX
X XX XXXX XX
X   XXXXXXEX
XXXXXXXXXXXX

Long path: [(0, 0), (0, 1), (1, 1), (0, 2), (0, 3), (0, 4), (0,
5), (0, 6), (0, 7), (0, 8), (0, 9), (0, 8), (0, 7), (0, 8), (0,
9), (0, 8), (0, 9), (1, 9), (0, 8), (0, 7), (0, 8), (0, 7), (0,
8), (0, 9), (1, 9), (2, 9), (3, 8), (2, 9), (1, 9), (0, 8), (0,
9), (0, 8), (0, 7), (0, 8), (0, 7), (0, 6), (0, 7), (0, 8), ...,
(7, 7), (8, 8), (7, 7), (8, 8), (9, 9)]
Short path: [(0, 0), (0, 1), (1, 1), (2, 2), (3, 3), (4, 4), (5,
5), (6, 6), (7, 7), (8, 8), (9, 9)]
Length of long path: 452.
Length of short path: 11.
```

Varje mellanslag representerar en plats som man kan gå på, och varje X representerar en vägg i labyrinten, och bokstaven E representerar utgången som vi vill hitta.

Nu ska du, givet denna kod, skriva klart ett program (bestående av funktioner och klasser) som löser en given labyrint med slumpvandringar (slumpmässiga förflyttningar mellan framkomliga rutor tills utgången hittas).

Som ett sista steg, ska den funna vägen genom labyrinten förfinas till en kortare väg (men inte nödvändigtvis den kortast möjliga vägen).

B1. Skriv metodkroppen för metoden `get_neighbours` som returnerar en lista innehållandes de 8 närmaste grannarna i form av tupler `[(x, y, value), (x, y, value), ..., (x, y, value)]` för varje närliggande ruta kring en ruta med given koordinat x, y genom att använda den givna metoden `get_neighbour`.

B2. Skriv klassdefinitionen för klassen `RandomMazeSolver`, inklusive konstruktorn, som mrepresenterar en navigerande agent som försöker finna en väg från början till slutet, och som fungerar med redan given kod. (Notera vad som skickas in som argument i koden där lösarobjektet skapas). Betänk vilka attribut som den kommer behöva. Den ska lagra den vandrande vägen som ett attribut, och den nuvarande platsen, och en referens till Maze-objektet.

B3. Skriv metoden `step` i klassen `RandomMazeSolver`, som tar ett slumpmässigt steg i labyrinten från agentens aktuella position. Använd metoden `get_neighbours` från klassen `Maze` (B1) och funktionen `random_shuffle` (från del A) på grann-listan för att realisera slumpmässighet. Steg ska bara tas till tomma platser, och till utgången (inte in i väggar). Om agenten anländer till utgången efter ett taget steg ska metoden returnera `True`, annars `False`. Varje ny position som besöks ska läggas till som en tupel (x, y) till `path`-attributet (en lista).

B4. Skriv en funktion shorten_path som tar en väg (path, en lista av tupler (x, y)) och beräknar en kortare väg baserat på den givna vägen, genom att ta bort alla genvägar (en del av en väg som inte ledde framåt mot utgången). En omväg kan defineras som en delväg (subpath) som startar på en given position och slutar på samma position (en slinga).

Exempel: I vägen [(1, 1), (2, 2), (2, 3), (2, 2), (3, 2), (3, 3)], är delvägen [(2, 2), (2, 3), (2, 2)] en omväg, eftersom vi kan välja vägen [(1, 1), (2, 2), (3, 2), (3, 3)], och hamna på samma position, men med färre steg.

`shorten_path` funktionen kan implementeras på flera sätt. Ett förslag är att använda följande algorithm:

Beskrivning av SHORTEN PATH algoritmen
Initialisera ett index `i` som indexet för det sista element i `path`-listan.
Skapa en ny tom `path`-lista.
Gör sedan följande tills `i` understiger 0:
Hitta det första indexet `j` i den ursprunliga `path`-listan som innehåller samma position som den som finns på den aktuella (i:de) positionen `path[i`. Lägg till `path[i]` sist i den nya `path`-listan.
Tilldela `i` att vara `j-1`.
Vänd ordning på den konstruerade `path`-listan (eftersom du la till slutet sist) och returnera resultatet.