Tentamen Programmeringsteknik I (Python) 2021-10-21

Lärare: Martin Almquist, Hans Karlsson
Skrivtid: 08:00 - 13:00

Skrivningen består av två delar. Lösningarna till uppgifterna på A-delen ska skrivas in i de tomma rutorna och den delen ska lämnas in. Rutorna är tilltagna i storlek så att de ska rymma svaren. En stor ruta betyder inte att svaret måste vara stort! Lösningarna till uppgifterna på B-delen skrivs på lösa papper.

Varje uppgift A1-A10 och B1-B3 blir antingen godkänd eller underkänd. En lösning måste inte nödvändigtvis vara helt korrekt för att uppgiften ska bli godkänd. Mindre fel kan tillåtas.

Betygskriterier

- För betyg 3 krävs minst 7 av 10 godkända uppgifter på A-delen.
- För betyg 4 krävs betyg 3 samt minst 2 av 3 godkända uppgifter på B-delen.
- För betyg 5 krävs betyg 3 samt 3 av 3 godkända uppgifter på B-delen.
- Observera att B-delen inte rättas om inte A-delen är godkänd.

Tänk på följande

- Skriv läsligt. Använd inte rödpenna.
- Skriv bara på framsidan av varje papper.
- Lägg uppgifterna i ordning. Skriv uppgiftsnummer (gäller B-delen) och din anmälningskod överst i högra hörnet på alla papper.
- Fyll i försättsbladet ordentligt.
- På B-delen är det tillåtet att införa hjälpfunktioner/hjälpmetoder och hjälpklasser. Uttrycket *skriv en funktion/metod* betyder alltså *inte* att lösningen inte får struktureras med hjälp av flera metoder eller funktioner.
- Du behöver inte skriva import-satser för att använda funktioner och klasser från standardbiblioteken.
- Alla uppgifter gäller programmeringsspråket Python och programkod skall skrivas i korrekt Python. Koden ska vara läslig med lämpliga variabelnamn och korta men beskrivande namn på funktioner, klasser och metoder. Glöm inte indentering! Betyget påverkas negativt av icke-lokala variabler, onödiga variabler, dålig läslighet och upprepning av identisk kod.

Lycka till!
Martin och Hans

Del A

A1. Denna uppgift består av 4 st flervalsfrågor. Kryssa i rutan bredvid rätt alternativ. För godkänt resultat totalt på uppgiften krävs rätt svar på minst 3 av 4 flervalsfrågor.

i) Vad skrivs ut av följande kod?

```
lst1 = [1, 3, 5]
lst2 = lst1 + lst1
lst2.append(4)
print(lst2[-2:])
```

a) `[4, 10]` ☐
b) `[1, 3]` ☐
c) `[4, 5]` ☐
d) `[4, 1]` ☐
e) `[5, 4]` ☐

ii) Vad är `math` i koden nedan?

```
import math
theta = 0.1
y = math.sin(theta)
```

a) En funktion ☐
b) En variabel ☐
c) En parameter ☐
d) En modul ☐
e) En klass ☐
f) Det går inte att avgöra ☐

iii) Vad är det för skillnad på uttrycken `2` och `"2"`?

a) Det första är av typen `float` och det andra av typen `str`. ☐
b) Det första är av typen `int` och det andra av typen `float`. ☐
c) Det första är av typen `int` och det andra av typen `str`. ☐
d) Det första är av typen `str` och det andra av typen `dict`. ☐
e) Det första är av typen `tuple` och det andra av typen `list`. ☐

iv) Vilken datatyp resulterar följande uttryck i?

```
[1, 2.0, 'a', 4][-2]
```

a) `int` ☐
b) `float` ☐
c) `str` ☐
d) `dict` ☐
e) `list` ☐

A2. Skriv en funktion `vector_add(tpl1, tpl2)` som adderar två vektorer i $\mathbb{R}^2$. Vektorerna representeras här av tupler med två element. Koden

```
p1 = (1, 2)
p2 = (3, 4)
p = vector_add(p1, p2)
print(f'{p1} + {p2} = {p}')
```

ska ge utskriften

```
(1, 2) + (3, 4) = (4, 6)
```

```
def vector_add(tpl1, tpl2):
   return (tpl1[0] + tpl2[0], tpl1[1] + tpl2[1])
```

A3. Skriv en funktion `between(lst, lower, upper)` som givet en lista med tal returnerar en ny lista med alla tal som är större än eller lika med `lower` och mindre än eller lika med `upper`. Koden

```
lst = [1, 2, 3, 4, 5, 4, 3, 2, 1]
print(between(lst, 1, 5))
print(between(lst, 2, 4))
```

ska ge utskriften

```
[1, 2, 3, 4, 5, 4, 3, 2, 1]
[2, 3, 4, 4, 3, 2]
```

```
def between(lst, lower, upper):
   return [x for x in lst if lower <= x <= upper]
```

A4. Skriv en funktion `n_smallest(lst, n)` som givet en lista med tal returnerar en ny lista med de `n` minsta talen. Koden

```
lst = [4, 5, 3, 1, 3, 7, 2]
print(n_smallest(lst, 2))
print(n_smallest(lst, 3))
print(n_smallest(lst, 4))
```

ska ge utskriften

```
[1, 2]
[1, 2, 3]
[1, 2, 3, 3]
```

Tips: Funktionen `sorted(lst)` returnerar en sorterad version av listan `lst`. Tal sorteras i stigande ordning.

```
def n_smallest(lst, n):
    lst_sorted = sorted(lst)
    return lst_sorted[0:n]
```

A5. Skriv en funktion `word_count(s)` som givet en sträng returnerar ett lexikon med alla ord i strängen som nycklar och antal förekomster som värden. Koden

```
text = 'O Captain! My Captain!'
d = word_count(text)
print(d)
```

ska ge utskriften

```
{'o': 1, 'captain': 2, 'my': 1}
```

Notera att alla ord har gjorts om till enbart gemener (små bokstäver).

Tips 1: Anropet `re.findall(r'[a-zA-Z]+', s)` returnerar en lista med alla ord i strängen `s`. Du kan anta att modulen `re` redan finns importerad i den kontext där du definierar din funktion.

Tips 2: Sträng-metoden `lower()` returnerar en likadan sträng, fast med versaler utbytta mot gemener.

```
import re
def word_count(s):
   word_list = re.findall(r'[a-zA-Z]+', s)
   d = {}
   for word in word_list:
      w = word.lower()
      if w in d:
         d[w] += 1
      else:
         d[w] = 1
   return d
```

A6. Skriv en funktion `print_word_count(d)` som givet ett lexikon skapat av `word_count` (se föregående uppgift) skriver ut alla ord och antal förekomster på ett prydligt sätt, beskrivet nedan. Inga överflödiga parenteser eller hakparenteser ska skrivas ut. Orden ska skrivas ut i bokstavsordning.

Exempel: Koden

```
text = 'O Captain! My Captain!'
d = word_count(text)
print_word_count(d)
```

ska ge utskriften

```
captain: 2
my: 1
o: 1
```

Tips 1: Anropet `list(d.items())` returnerar en lista med tupler av alla nyckel-värde-par i lexikonet `d`. Varje tupel innehåller nyckeln som första element och värdet som andra element.

Tips 2: List-metoden `sort()` sorterar en lista av tupler baserat på tuplernas första element.

OBS! Du behöver inte ha skrivit funktionen `word_count` för att kunna lösa den här uppgiften. Du kan anta att `word_count` finns implementerad.

```
def print_word_count(d):
   lst = list(d.items()) # List of tuples of the form (key, value)
   lst.sort() # Sort in alphabetical order
   for tpl in lst:
      print(f'{tpl[0]}: {tpl[1]}')
```

A7. Skriv en funktion `rolls_required(value)` som simulerar att en tärning kastas tills den visar en siffra som är större än eller lika med heltalet `value`. Funktionen ska returnera antalet kast som krävdes innan tärningen visade en tillräckligt stor siffra.

Tips 1: Använd anropet `random.randint(1, 6)` för att slumpa fram ett heltal mellan 1 och 6 som representerar resultatet av ett tärningskast. Du kan anta att modulen `random` redan finns importerad i den kontext där du definierar din funktion.

OBS! Din funktion ska kunna returnera olika antal kast vid olika anrop med samma argument.

Koden

```
val = 5
num_rolls = rolls_required(val)
print(f'{num_rolls} rolls required to obtain a value >= {val}.')
num_rolls = rolls_required(val)
print(f'{num_rolls} rolls required to obtain a value >= {val}.')
```

ska *exempelvis* kunna ge utskriften

```
4 rolls required to obtain a value >= 5.
1 rolls required to obtain a value >= 5.
```

```
import random
def rolls_required(value):
   roll = random.randint(1, 6)
   count = 1
   while roll < value:
      roll = random.randint(1, 6)
      count += 1
   return count
```

A8. Skriv en funktion `dot_product(u, v)` som beräknar skalärprodukten av två vektorer med n element (här representerade med listor). Skalärprodukten $\mathbf{u} \cdot \mathbf{v}$ är definierad som

$$\mathbf{u} \cdot \mathbf{v} = u_0 v_0 + u_1 v_1 + \ldots + u_{n-1} v_{n-1}.$$

Koden

```
u = [1, 2]
v = [3, 4]
u_dot_v = dot_product(u, v)
print(f'The dot product of {u} and {v} equals {u_dot_v}.')
```

ska ge utskriften

```
The dot product of [1, 2] and [3, 4] equals 11.
```

Funktionen `dot_product(u, v)` ska tillåta listor av godtycklig längd som argument, men du kan anta att de två listorna `u` och `v` är lika långa.

```
def dot_product(u, v):
    res = 0
    for i in range(len(u)):
        res += u[i]*v[i]
    return res
```

A9. Betrakta klassen `Employee` nedan. Objekt av klassen `Employee` karakteriseras av ett namn och en adress. Än så länge finns bara en `__init__`-metod i klassen.

```
class Employee:
   def __init__(self, name, address):
      self.name = name
      self.address = address
```

Skriv en metod i klassen `Employee` som gör att koden

```
e1 = Employee('Martin', '1 Python Drive')
print(e1.address)
e1.move('2 Loop Road')
print(e1.address)
```

resulterar i utskriften

```
1 Python Drive
2 Loop Road
```

```
def move(self, new_address):
      self.address = new_address
```

A10. Skriv ytterligare en metod i klassen `Employee`. När metoden är implementerad ska koden

```
e1 = Employee('Martin', '1 Python Drive')
e2 = Employee('Hans', '3 Tuple Way')
print(e1)
print(e2)
```

resultera i utskriften

```
Martin lives on 1 Python Drive.
Hans lives on 3 Tuple Way.
```

OBS! Du behöver inte ha löst föregående uppgift för att lösa den här uppgiften.

```
def __str__(self):
   return f'{self.name} lives on {self.address}.'
```

Del B

Här ska du simulera smittspridning. Simuleringen kommer bestå av ett antal personer som rör sig slumpmässigt inom en kvadrat. Om en frisk person kommer nära en smittad person finns det en viss risk att den friska personen smittas. Efter en viss tid tillfrisknar smittade personer. Dessa kan då återigen smittas om de kommer nära en smittad person. Simuleringen kommer att bygga på klassen `Person`, vars konstruktor finns nedan.

```
import random
import math

class Person:
   def __init__(self, is_infected, recovery_time, box_size):

      self.is_infected = is_infected
      self.recovery_time = recovery_time
      self.time_since_infection = 0
      self.box_size = box_size

      x = box_size*random.random()
      y = box_size*random.random()
      self.pos = (x, y)
```

Klassens instansvariabler är tänkta att användas på följande sätt:

- `is_infected`: `bool`. `True` om personen just nu är smittad, annars `False`.
- `recovery_time`: `int`. Tid det tar för personen att bli frisk efter att hen blivit smittad.
- `time_since_infection`: `int`. Tid som passerat sedan personen blev smittad. Om personen just nu är frisk har variabelns värde ingen mening.
- `box_size`: `int` eller `float`. Sidlängd på den kvadrat som personen rör sig inuti.
- `pos`: `tuple`. Personens nuvarande koordinater.

Utöver konstruktorn finns än så länge två metoder i klassen: `step` och `advance_time`. Metoden `step` förflyttar personen i en slumpmässig riktning. Förflyttningens längd överstiger inte `max_length`. Om personens nya position är utanför kvadraten slumpas ett nytt steg tills den nya positionen är innanför kvadraten. Nedan visas av utrymmesskäl bara en del av metoden.

```
def step(self, max_length):

   # More code here.

   self.pos = (x_new, y_new)
```

Metoden `advance_time` kan sägas stega fram personens inre klocka. Så länge personen är frisk gör metoden ingenting. Om personen däremot är smittad ökar metoden på `time_since_infection` med 1. Om personen har varit smittad `recovery_time` tidsenheter blir hen frisk, det vill säga `is_infected` tilldelas värdet `False` och `time_since_infection` nollställs.

```
def advance_time(self):
   if self.is_infected:
      self.time_since_infection += 1
      if self.time_since_infection == self.recovery_time:
         self.is_infected = False
         self.time_since_infection = 0
```

B1. Skriv en metod `distance(self, other)` som beräknar och returnerar avståndet till ett annat objekt av klassen `Person` (parametern `other`).

```
def distance(self, other):
      dx = self.pos[0] - other.pos[0]
      dy = self.pos[1] - other.pos[1]
      return (dx**2 + dy**2)**0.5
```

B2. Skriv en metod `handle_proximity(self, other)` som ska anropas om personen befunnit sig nära en annan person (`other`). Metoden ska utföra följande:

- Om `self` är smittad, men inte `other`, ska `other` smittas med 50% sannolikhet.
- Om `other` är smittad, men inte `self`, ska `self` smittas med 50% sannolikhet.
- Om båda är friska eller båda är smittade ska inget utföras.

OBS! Metoden behöver inte kontrollera huruvida personerna är nära varandra. Detta görs på annan plats i koden.

```
def handle_proximity(self, other):

   if other.is_infected and not self.is_infected:
      # Check if self was infected by other
      x = random.random()
      if x < 0.5:
         self.is_infected = True

   if self.is_infected and not other.is_infected:
      # Check if other was infected by self
      x = random.random()
      if x < 0.5:
         other.is_infected = True
```

B3. Nedan finns första delen av en `main`-funktion som ska finnas i samma modul som klassen `Person`. Koden definierar ett antal parametrar och skapar sedan `persons`, som är en lista med `N` objekt av klassen `Person`.

```
def main():

   # --- Simulation parameters --------------------------------
   N = 40                          # Number of people
   Nsteps = 100                    # Number of time steps
   box_size = 10                   # Size of simulation area
   max_step = 1                    # Longest possible step
   initial_infection_prob = 0.5    # Initial infection probability
   recovery_time = 10              # Time to recover after infection
   infection_radius = 0.5          # Infected individual can only infect
                                   # others if they are within this distance
   #----------------------------------------------------------

   #--- Create list of Person objects ----------------------
   persons = []
   for _ in range(N):
      is_infected = random.random() < initial_infection_prob
      p = Person(is_infected, recovery_time, box_size)
      persons.append(p)
   #----------------------------------------------------------
```

Skriv färdigt `main` genom att fylla på med kod *under* den givna koden. Din kod ska simulera `Nsteps` stycken tidssteg. I varje tidssteg ska:

1. alla personers inre klocka stegas fram med metoden `advance_time`.
2. alla personer ta ett slumpmässigt steg, med maximal längd `max_step`.
3. alla personer kontrollera om de befinner sig inom avståndet `infection_radius` från någon annan person. Om så är fallet ska metoden `handle_proximity` anropas. Om en av personerna är smittad kommer den andra då att smittas med viss sannolikhet. Notera att du här inte behöver ha skrivit metoderna `distance` eller `handle_proximity` utan kan anta att de finns implementerade.

```
for _ in range(Nsteps):
      for p in persons:
         p.advance_time()
      for p in persons:
         p.step(max_step)
      for i in range(N):
         for j in range(i+1, N):
            if persons[i].distance(persons[j]) < infection_radius:
               persons[i].handle_proximity(persons[j])
```