

## Tentamen Programmeringsteknik I 2020-03-20

Skrivtid: 08:00-13:00. 20 minuters extra marginal för elektronisk inlämning.

### Tänk på följande

- Om du var anmäld till salstentan anger du din anmälningskod på första svarssidan (ovanför uppgift A1).
- Om du laddar upp i Studentportalen anger du din anonymkod i fältet under. (Du kan alltså ange båda.)
- Ladda i första hand upp din besvarade tenta i Studentportalen. Maila i andra hand ifylld PDF till `carl.nettelblad@it.uu.se`. Om du inte lyckas fylla i PDF-filen, skriv svar på uppgifterna *i ordning med tydlig markering av varje deluppgift* i en textfil eller ett Worddokument. I sista hand kan avfotograferade eller scannade lösningar på papper godtas.
- Såvida inget annat anges får man bygga på lösningar till föregående uppgifter även om dessa inte har lösts.
- Det är tillåtet att införa extra metoder eller funktioner. Uttryck som ”skriv en funktion som” skall alltså *inte* tolkas så att lösningen inte får struktureras med hjälp av fler funktioner.
- Alla uppgifter gäller programmeringsspråket Python och programkod skall skrivas i Python. Koden skall vara läslig dvs den skall vara vettigt strukturerad och indenterad. Namn på variabler, funktioner, metoder, klasser etc skall vara beskrivande men kan ändå hållas ganska korta.

Observera att betyget kan påverkas negativt bland annat av:

- onödiga variabler,
- dålig läslighet,
- upprepning av identisk kod,
- underlåtenhet att utnyttja given eller egen tidigare skriven kod,
- hög grad av ineffektivitet, som att upprepa ett omfattande funktionsanrop ett potentiellt stort antal gånger.

### Observera

**Skrivningen består av två delar. Lösningarna till uppgifterna på del A har svarsrutor i direkt anslutning.** Rutorna är tilltagna i storlek så att de ska rymma svaren. Om du ändå inte får plats finns det extra tomsidor i slutet. Markera gärna i så fall att dessa använts.

Lämna rättningsrutorna tomma! För A1 finns extra utrymme att kommentera dina kryssvar om du vill förklara något.

**Svaren på del B skrivs ett per speciellt anvisad sida, efter uppgiftstexterna.**

Det är ditt ansvar att tentan blir elektroniskt inlämnad med svar på alla uppgifter du önskar besvara. Kontrollera att det verktyg du använder klarar av att spara ifyllda PDF-formulär, så du inte bara får en tom fil. Adobe Acrobat Reader klarar garanterat att spara och läsa våra filer. Se till att spara ditt formulär ofta och kontrollera på slutet att du verkligen laddar upp rätt fil!

*Var god vänd!*

## Bedömning och betygsättning

För att bli godkänd (betyg 3) krävs att A-delen i huvudsak är rätt löst. Det betyder inte att varje uppgift behöver vara exakt rätt, men att du ska visa att du i huvudsak uppfyller kursens mål, som säger att studenten efter godkänd kurs ska kunna:

- redogöra för de grundläggande begreppen modul, funktion, klass, objekt och därtill hörande underbegrepp
- analysera och lösa problem med hjälp av programmeringskonstruktioner
- förklara vad ett givet program i Python utför
- använda befintliga moduler och skriva program med flera samverkande komponenter i Python
- använda en programutvecklingsmiljö
- testa och felsöka program

För betyget 4 krävs dessutom att minst hälften, och för betyg 5 alla, uppgifterna på B-delen är i stort sett rätt lösta. Vid bedömning för dessa betyg tas även hänsyn till kvaliteten på lösningarna på A-delen.

Observera att B-delen normalt sett endast rättas om A-delen är godkänd.

## Hjälpmedel

En del viktiga aspekter av Python tas upp på separat referensblad.

Eftersom detta är en hemtenta tillåts du använda tillgängliga elektroniska och fysiska resurser. Det innefattar att köra kod i Python och söka i dokumentation, kurshemsidor och liknande resurser på nätet. Det innefattar inte att ställa frågor eller på andra sätt kommunicera med andra människor. Tentamen ska göras enskilt. Om du utgår från ett specifikt exempel i din kod bör du ange källan till detta. Kopiering av kod utan att ange källa kan komma att betraktas som plagiat – utgångspunkten är att du ska formulera dina egna svar i sin helhet.

Om du inte minns exakt vad en viss funktion heter eller hur en del av språkets syntax ser ut kan du påpeka detta och beskriva vilka antaganden du gör. Vi försöker både bedöma hur väl du har löst problemet och hur väl du kan hantera Pythons språk och funktionalitet.

Lycka till!

Anmälningsskod:

Anonymkod:

## Del A (obligatorisk för alla)

A1. Ringa in rätt alternativ, endast ett om frågan inte uttryckligen nämner att flera kan vara möjliga.

- a) Vilken funktion i modulen `math` räknar ut den euklidiska längden på en vektor som lagras som en lista?  
Se <https://docs.python.org/3/library/math.html>
- 1) `ceil`  
2) `comb`  
3) `dist`  
4) `erf`  
5\*) `hypot`  
6) `lgamma`
- b) Vad är sant om en tupel?
- 1) Alla element måste ha samma typ  
2) En tupels ingående värden/referenser kan bara anges när den skapas  
3) En tupel har alltid mer än ett element  
4) Om man gör `print` på en tupel får man alltid samma resultat eftersom den är oföränderlig  
5) Varje sträng är också en tupel
- c) Vad är `random` i följande kod?
- ```
import random  
  
x = random.randint(0, 100)
```
- 1) En funktion  
2) En variabel  
3) En parameter  
4) En modul  
5) En klass  
6) Det går inte att säga
- d) Vad är `Worker` i följande kod?
- ```
x = Worker  
y = x(999)
```
- 1) En funktion  
2) En variabel  
3) En parameter  
4) En modul  
5) En klass  
6) Det går inte att säga
- e) Vad kallar man av konvention den första parametern till en metod?
- 1) `me`  
2) `__init__`  
3) `this`  
4) `self`  
5) `append`  
6) `0`

- f) Om följande rad finns i ett program och inte får ändras, vad är viktigast att se till också finns i programmet?
- ```
file = open('ourfile.txt', 'r')
```
- 1) `file.close()`  
2) `text = file.readlines()`  
3) `print(file)`  
4) `for c in file.read():`  
5) `with file.readlines() as lines:`
- g) Vad är skillnaden på en funktion och en metod?
- 1) Man kan själv definiera egna funktioner  
2) Man kan själv definiera egna metoder  
3) Alla metoder är magiska  
4) En funktion anropas på ett objekt  
5) En metod anropas på ett objekt  
6) Bara funktioner kan modifiera objekt  
7) Bara metoder kan modifiera objekt
- h) Hur många olika listor skapar följande kod?
- ```
a = [1, [2, 3]]  
b = a  
c = a[1]  
d = a[1][0]  
e = a[1][1]  
print(a.pop)
```
- 1) 1  
2) 2  
3) 3  
4) 4  
5) 5
- i) Vilken typ har följande uttryck?
- ```
[1, 2, 3.5, 4][2]
```
- 1) int  
2) float  
3) str  
4) list

- a Det avsedda svaret är `hypot(*vektor)`, vilket är en lite ovanlig syntax. På grund av detta har vi även givit rätt för `dist`, trots att man då måste fylla på med en extra nollvektor själv.
- h Det skapas bara två listobjekt. Övriga är referenser till dessa. `pop` är inget anrop, det är en referens till den metoden (eftersom parenteser saknas). Man kan visualisera förloppet på exempelvis [pythontutor.com](http://pythontutor.com) för att se detta om det inte framgår intuitivt.

- A2. Funktionen `cube(lst)` ska ta emot en lista som parameter och returnera en ny lista med varje tal taget i kubik (upphöjt till tre). Skriv den nödvändiga koden för funktionen, samt ett exempel som använder funktionen för att beräkna och skriva ut kuberna för talen 5, 8 och 9.

```
def cube(lst):
    """Return an output list with the cube of each element in the input."""
    return [i**3 for i in lst]

print(cube([5, 8, 9]))
```

- A2F1 Itererar över index i stället för element där det inte behövs. I en Pythonloop försöker vi iterera över det vi faktiskt arbetar med, vilket ofta blir element i listorna i stället för `range(len(lst))` eller liknande.
- A2F2 Koden demonstrerar inte hur funktionen ska användas genom ett anrop till funktionen `cube`.
- A2F3 Demonstrationskoden skriver inte ut resultatet. Att bara anropa `cube` leder inte till utskrift.
- A2F4 Koden returnerar inte en ny lista, utan förändrar den befintliga. Jämför med uppgifterna i OU2.
- A3. Funktionen `capitalize` ska ta emot ett ord som parameter i form av en sträng och returnera en version där det första tecknet skrivs med versal. Övriga tecken ska vara oförändrade. Exempelvis ska 'hej' bli 'Hej'. Skriv klart funktionen. Tänk på att `str` har en metod `upper` som returnerar en ny sträng där alla tecken är versaliserade.

```
def capitalize(word):
```

```
    return word[0].upper() + word[1:]
```

- A3F1 Funktionshuvudet var givet i uppgiften. Den givna lösningen ger ett nytt, förändrat, funktionshuvud.
- A3F2 Lösningen returnerar `None`.
- A3F3 Lösningen splittar strängen i många delar och förenar med `join`. Detta är onödigt.
- A3F4 Lösningen innehåller en onödig loop.
- A3F5 Lösningen saknar retursats.
- A3F6 En funktion ska normalt sett inte skriva ut något om det inte direkt har anvisats. Du vet inte i vilket sammanhang funktionen ska användas.
- A3F7 Lösningen använder inte inparametern till funktionen.
- A3F8 Lösningen skapar inte ett korrekt svar.
- A3F9 Lösningen använder f-sträng på värden som redan är strängar utan andra formateringsbehov.
- A3F10 Lösningen använder `join` på ett ineffektivt sätt.
- A3F11 Lösningen versaliserar alla förekomster av första bokstaven.
- A3F12 Lösningen versaliserar alla bokstäver och plockar sedan ut de önskade.
- A3F13 Lösningen omdefinierar metodnamnet, som var givet till `capitalize`.

A4. Nedan finns innehållet i en funktion. Föreslå hur början av funktionsdefinitionen (dess "huvud") ska se ut med namn och parametrar. Skriv även en lämplig docstring.

```
def countQ(line):  
    """Count the number of occurrences of the letter Q in input string."""  
  
    count = 0  
    for c in line:  
        if c == 'Q':  
            count += 1  
    return count
```

- A4F1 Lösningen saknar helt def-rad för funktionen (funktionshuvudet).
- A4F2 Lösningen saknar helt docstring.
- A4F3 Utifrån koden framgår att parametern heter `line`. Ett annat namn anges i funktionshuvudet.
- A4F4 En docstring är en sträng, som skrivs separat, först i funktionen. Det är inte en kodkommentar med `#`.
- A4F5 En docstring är av konvention en multiline-sträng, d.v.s. tre citationstecken eller möjligen tre apostrofer i följd. Du har använt en annan sträng.
- A4F6 Din docstring är inte korrekt indenterad relativt funktionshuvudet.
- A4F7 En docstring ska finnas omedelbart under funktionshuvudet, korrekt indenterat. Du har placerat den på något annat ställe.
- A4F8 Ett funktionsnamn ska beskriva vad funktionen gör. Här bör det framgå att den räknar antalet `Q`. `countQ`, `Qcounter`, `howmanyQ`, `num_of_Q_in_str` vore exempel på någorlunda tänkbara namn.
- A4F9 En docstring ska beskriva vad funktionen gör, inte bara vad den heter eller vad den tar för parametrar, det syns i funktionshuvudet. Den ska heller inte bara beskriva rad för rad vad koden gör, det syns i koden. Exempel: `"""Sätter count till 0, loopar över varje tecken c i line och ökar count ibland. Returnerar sedan count."""` duger inte, eftersom strängen då beskriver allt som händer i funktionen, utom vad den faktiskt har för syfte.

A5. Funktionen `longestWord` ska returnera det längsta ordet från en sträng. Tänk på att metoden `split` kan användas för att dela upp en sträng i ord. Skriv innehållet till funktionen.

```
def longestWord(data):  
    """Returns the longest word found in input string data. Words are  
    defined as being delimited by whitespace."""
```

```
        sofar = ''  
        for word in data.split():  
            if len(word) > len(sofar):  
                sofar = word  
  
        return sofar
```

- A5F1 Funktionen ska returnera resultatet, inte skriva ut det.
- A5F2 Algoritmen är klart mer komplicerat skriven än den behöver vara.
- A5F3 Strängen delas aldrig upp i ord.
- A5F4 Det angavs uttryckligen i docstring att ett ord definierades som text som omges av tomrum, inte bokstäver specifikt. Lösningen använder en annan definition.
- A5F5 Fel information returneras, exempelvis att längden på det längsta ordet returneras i stället för ordet i sig.
- A5F6 Koden använder `sort`, `sorted`, `max` eller liknande, men anger ingen `key`, så orden sorteras i (ungefär) alfabetisk ordning i stället för efter längd.

- A6. Definiera en ny klass `Point` som beskriver en koordinat i ett tvådimensionellt rum. Dess initieringsmetod ska ta in två parametrar `x` och `y` och internt lagra koordinaterna som en tupel i ett attribut (en instansvariabel) `pos`. Det ska inte finnas några andra attribut.

```
class Point:
    def __init__(self, x, y):
        self.pos = (x, y)
```

- A6F1 Din funktion skapar ytterligare attribut än `pos`, t.ex. `self.x`.
- A6F2 Koordinaterna ska anges som parametrar och lagras i `pos` i ordningen `x, y`.
- A6F3 Initieringsmetoder förväntas inte returnera något.
- A6F4 `self.pos` ska initieras till tupeln `(x,y)`.

- A7. Skriv en ny metod till `Point`, `rot90`. `rot90` ska vrida den befintliga punkten i steg om 90 grader moturs. `p.rot90()` ska då vrida 90 grader moturs, `p.rot90(2)` ska vrida 180 grader moturs och både `p.rot90(3)` och `p.rot90(-1)` ska vrida 270 grader moturs (vilket även motsvarar 90 grader medurs).

Ledning: Rotation 90 grader moturs ( $\pi/2$  radianer) motsvarar multiplikation med  $i$  för komplexa tal. Annorlunda uttryckt kan de nya roterade koordinaterna  $(x_r, y_r)$  skrivas som:

$$x_r = -y$$
$$y_r = x$$

```
def rot90(self, n=1):
    for _ in range(n % 4):
        self.pos = (-self.pos[1], self.pos[0])
```

- A7F1 Flera 90-graderssteg än fyra bör hanteras genom modulo, inte genom väldigt lång loop.
- A7F2 Lösningen måste kunna hantera alla tänkbara 90-graderssteg, inte bara vissa fall.
- A7F3 Antalet rotationssteg ska ha standardvärdet 1 för att `p.rot90()` ska fungera.
- A7F4 Metoden ska inte returnera något.
- A7F5 Avrundning hjälper mot precisionsfel från trigonometri för heltalskoordinater, kan ge katastrofala fel för flyttalskoordinater.
- A7F6 `self.pos` måste uppdateras.
- A7F7 En metod måste ha `self`-argument.
- A7F8 Metoden ska inte skriva ut något.

- A8. Python använder magiska metoder med specifika namn för specifik funktionalitet. Skriv klart metoden `__eq__` för `Point`. `__eq__` används av Python när jämförelser utförs med `==`. Metoden ska returnera `True` om två punktobjekt har samma koordinater och `False` annars.

```
def __eq__(self, other):
    """Compares self to other, returning True if two Point instances
    represent the same coordinates, False otherwise."""
```

```
    return self.pos == other.pos
```

- A8F1 Metoden måste returnera resultatet av jämförelsen.
- A8F2 Jämförelsen ska ske mellan `self.pos` och `other.pos`.
- A8F3 Metoden måste returnera `False` om jämförelsen misslyckas. Denna lösning returnerar inte, d.v.s. får returvärdet `None`.

A9. Antag nu att all kod du har skrivit i `Point` finns i filen `smallvector.py`. Skriv den nödvändiga koden för att använda `Point` från en annan Pythonfil, skapa två vektorer med koordinaterna  $x_1 = 3, y_1 = 5; x_2 = -5, y_2 = 3$ , rotera den andra 270 grader moturs, jämföra dem med `==` och skriva ut resultatet.

```
from smallvector import Point

p1 = Point(3, 5)
p2 = Point(-5, 3)
p2.rot90(3)
print(p1 == p2)
```

- A9F1 Den andra punkten ska roteras 3 eller -1 90-graders steg moturs med `rot90`.
- A9F2 Punkterna ska jämföras, A9 ska inte titta på punkternas `pos`.
- A9F3 Klassen `Point` ska användas i A9.
- A9F4 `smallvector` måste importeras på något sätt.
- A9F5 Resultatet av jämförelsen ska skrivas ut med `print`.
- A9F6 `rot90` ska inte returnera, utan förändrar ett befintligt objekt.

A10. Nedanstående program innehåller ett antal mindre skrivfel. Skriv den korrekta koden med korta kommentarer om vad som var fel. Funktionen `type` returnerar typen för det värde som anges som parameter.

```
lex = {}
lex{'a'} = 'hej'
lex['b'] = [5 9]
for k in lex.keys:
    print('Information for key {k}')
    print f'Val: {lex[k]}, Type: {type(lex[k])}'
```

```
lex = {}
lex['a'] = 'hej' # Hakklammer för indexering
lex['b'] = [5, 9] # Komma mellan element
for k in lex.keys(): # () i anrop till keys
    print(f'Information for key {k}') # Prefix f för formatsträng
    print(f'Val: {lex[k]}, Type: {type(lex[k])}') # Parenteser i anrop
```

- A10F1 På den första `print`-raden behövs ett `f` för att det ska vara en formatsträng. Det finns inget skäl att tro att texten `{k}` skulle skrivas ut oförändrad.
- A10F2 Kommentarer om vad som var fel saknas i det givna svaret.



## Del B (för betyg 4 och 5)

Svaren till dessa uppgifter ges på separata svarssidor längst bak. Se till att skriva varje uppgift på anvisat blad.

- B1. Skriv innehållet till nedanstående funktion. Den ska bygga upp ett lexikon med nycklar och värden. För varje sådan post ska den läsa in två rader från tangentbordet. Den första raden ska utgöra nyckeln som ska lagras i en form med enbart versaler. Användaren skriver in värdet på nästa rad. Detta ska lagras som ett heltal. Om man anger en tomrad som nyckel ska inläsningen avslutas och lexikonet returneras.

```
def read_uppercase_keys_and_ints():
    """Reads two lines per entry, one with key, one with value, stores all
    entries in a dictionary, which is returned. Keys are stored as UPPERCASE only,
    values as ints. Input is stopped by entering an empty line as key."""

    d = {}
    while True:
        key = input('Nyckel: ').upper()
        if key == '':
            return d
        d[key] = int(input('Värde (heltal): '))
```

B1F1 Retursats saknas.

B1F2 Lösningen stöder inte att inmatning avslutas med tom rad som nyckel.

B1F3 Lösningen skriver ut lexikonet i stället för att returnera det från funktionen.

B1F4 Värdena lagras ej som typen int i lexikonet.

- B2. Antag att du har en lista `plist` där elementen är punkter i form av objekt av klassen `Point`, som du skrev i föregående tentamensdel. Vi vill nu sortera punkterna i *vinkelordning* (relativt origo  $(0,0)$ , med vinkel 0 grader/radianer pekande rakt åt höger  $(1,0)$  och positiva vinklar gående moturs). Resultatet av sorteringen, fortfarande objekt av klassen `Point`, ska lagras i `plist2`. Den ursprungliga listan ska inte förändras. Skriv kod som utför detta. Du kan utnyttja funktionen `atan2` i modulen `math`, som kan användas för att ta reda på vinklar enligt denna definition. Den har följande dokumentation:

**`math.atan2(y, x)`** Return `atan(y / x)`, in radians. The result is between  $-\pi$  and  $\pi$ . The vector in the plane from the origin to point  $(x, y)$  makes this angle with the positive X axis. The point of `atan2()` is that the signs of both inputs are known to it, so it can compute the correct quadrant for the angle. For example, `atan(1)` and `atan2(1, 1)` are both  $\pi/4$ , but `atan2(-1, -1)` is  $-3\pi/4$ .

```
import math

def getangle(p):
    return math.atan2(p.pos[1], p.pos[0])

plist2 = sorted(plist, key=getangle)
```

- B2F1 Överkomplicerad listhantering.
- B2F2 Om sorteringen görs i en funktion bör ett funktionsanrop med `plist` som parameter göras, och resultatet bör sparas i en variabel som heter `plist2`. Dessa namn angavs i uppgiften.
- B2F3 Användning av lexikon. `Point`-objekt som nyckel kan ge problem med exempelvis dubletter av punkter, vinkel som nyckel missar om flera punkter har samma vinkel.
- B2F4 Onödigt komplex lösning.
- B2F5 Inget skäl att skapa nya `Point`-objekt.
- B2F6 Resultatet blir en lista med vinklar inte `Point`-objekt.
- B2F7 `atan(y, x)` anropas fel.
- B2F8 Ett `Point`-objekt är inte en tupel. Koordinaterna finns i `p.pos`.
- B2F9 Den ursprungliga listan `plist` förändras.

B3. Återstående uppgifter handlar om två klasser, `FallingBall` och `RollingBall`. Dessa ska utföra en mycket enkel (direkt dålig) simulering med tidsstegning, som styrs av koden nedan. Din kod ska även fungera om konstanterna i exemplet nedan byts ut eller om man skapar flera objekt av samma klass.

```
balls = [FallingBall(10, 0, 0.5), RollingBall(10, 10, 0.2)]

timestep = 0.01
for i in range(0, 1000):
    time = i * timestep
    if i % 10 == 0 or i < 5:
        print(f'Time: {time:5.2f}')
        print(f'{balls}')

    for b in balls:
        b.step(timestep)
```

Inledningen för utmatningen från programmet ska se ut som nedan. Värdena från din lösning ska stämma exakt.

```
Time: 0.00
[FallingBall(10.0000, 0.0000), BouncingBall(10.0000, 10.0000)]
Time: 0.01
[FallingBall(9.9990, -0.0981), BouncingBall(10.0998, 9.9804)]
Time: 0.02
[FallingBall(9.9971, -0.1962), BouncingBall(10.1994, 9.9608)]
Time: 0.03
[FallingBall(9.9941, -0.2941), BouncingBall(10.2988, 9.9411)]
Time: 0.04
[FallingBall(9.9902, -0.3917), BouncingBall(10.3980, 9.9215)]
Time: 0.10
[FallingBall(9.9464, -0.9675), BouncingBall(10.9892, 9.8038)]
```

I vår enkla simulering har fallande bollar bara y-koordinater och rullande bollar (på en plan yta) bara x-koordinater. De utsätts för gravitation från jorden. I initieringsmetoden får båda en position (i y-led respektive x-led, positiva y uppåt, positiva x höger), en hastighet (i y-led respektive x-led) och en luftmotståndskoefficient respektive en friktionskoefficient.

Skriv början på definitionen av de båda klasserna med initieringsmetoder som lagrar nödvändig information i olika attribut.

```
class FallingBall:
    def __init__(self, y, v, airres):
        self.y = y
        self.v = v
        self.airres = airres
```

```

def __str__(self):
    return f'FallingBall({self.y:.4f}, {self.v:.4f})'

def __repr__(self):
    return self.__str__()

def step(self, timestep):
    a = -9.81
    aira = self.v * self.v * self.airres
    if self.v > 0:
        a -= aira
    else:
        a += aira

    self.v += a * timestep
    self.y += self.v * timestep

    if self.y < 0:
        self.y = -self.y
        self.v = -self.v

class RollingBall:
    def __init__(self, x, v, fric):
        self.x = x
        self.v = v
        self.fric = fric

    def __str__(self):
        return f'RollingBall({self.x:.4f}, {self.v:.4f})'

    def __repr__(self):
        return self.__str__()

    def step(self, timestep):
        a = 9.81 * self.fric * timestep
        if self.v > 0:
            self.v = max(0, self.v - a)
        else:
            self.v = min(0, self.v + a)

        self.x += self.v * timestep

balls = [FallingBall(10, 0, 0.5), RollingBall(10, 10, 0.2)]

timestep = 0.01
for i in range(0, 1000):
    time = i * timestep
    if i % 10 == 0 or i < 5:
        print(f'Time: {time:5.2f}')
        print(f'{balls}')

    for b in balls:
        b.step(timestep)

```

- B3F1 Öväntad typomvandling.
- B3F2 Överflödigt att beräkna accelerationen. Den ändras för varje tidssteg.
- B3F3 Attributtilldelningarna motsvarar inte parameternamn.
- B3F4 Minst en inparameter saknas.

B4. Skriv metoderna `__repr__` och `__str__` för båda klasserna. De ska returnera strängar som beskriver objektets tillstånd, med utseende motsvarande `FallingBall(y, v)` och `RollingBall(x, v)` där `x`, `y` och `v` är värden för det aktuella objektet. `__repr__` och `__str__` för respektive klass ska alltså returnera samma sak, men se till att inte duplicera kod i onödan.

- a) Skriv de båda metoderna för `FallingBall`.
- b) Skriv de båda metoderna för `RollingBall`.

- B4F1 Fel antal decimaler i utskriften.
- B4F2 Onödigt duplicerad kod. `__str__` kan anropa `__repr__` eller vice versa. Det går också att skriva en gemensam hjälpfunktion eller hjälpmetod som anropas av flera.

B5, B6. I följande två uppgifter ska vi implementera metoden `step` som givet ett tidssteg  $\Delta t$  (i sekunder) uppdaterar objektets hastighet och position för båda klasserna. Allmänt ska metoderna beräkna accelerationen  $a$  och uppdatera  $v$  enligt  $v_1 = v_0 + a\Delta t$ . Positionen ska sedan uppdateras enligt  $p_1 = p_0 + v_1\Delta t$ .

Det krävs dock vissa extra försiktighetssteg för att vår simulering inte ska bli orimlig med denna mycket enkla tidsstegning. När den fallande bollen når marken ska den, till synes momentant, studsas uppåt i stället, med hela farten bevarad (en så kallad elastisk stöt). Fart nedåt byter tecken till fart uppåt.

Den rullande bollen bromsas av friktion. Om man är oförsiktig kan det få hastigheten att byta tecken upprepade gånger. Se till att din boll i stället stannar helt och hållet om friktionen är tillräckligt stor för att föra hastigheten till 0 inom ett tidssteg.

Tänk på att även om du inte kan räkna igenom en lång tidsstegning för hand kan exemplet på utmatning ovan användas för att kontrollera delar av korrekt beteende, som positivt/negativt tecken på olika värden, ungefär i vilken storleksordning de ska vara, om de växer snabbare eller långsammare över tid o.s.v.

B5. Implementera `FallingBall.step`.

Det finns hela tiden en nedåtriktad (negativa  $y$ ) tyngdacceleration  $9.81\text{m/s}^2$ . Det betyder att farten nedåt under en sekund ökar med  $9.81\text{m/s}$ . På exempelvis  $0.1\text{s}$  ökar den med  $0.981\text{m/s}$ .

Dessutom finns ett luftmotstånd. Accelerationen från luftmotståndet i vår modell är beroende av dels *kvadraten* på den nuvarande hastigheten och dels en luftmotståndskoefficient, som vi angav i initeringsmetoden. Luftmotståndet är förstas alltid riktat åt motsatt håll relativt bollens nuvarande hastighet. När bollen är på väg uppåt bromsas den nedåt och vice versa.

Se till att din `step`-metod får bollen att studsas med bevarad fart på ett rimligt sätt varje gång den träffar  $y = 0$ . Motivera hur du gör detta.

Ledning: Förutom studsens kan accelerationen som verkar på bollen alltså beskrivas som  $a = -9.81 - fv^2 \text{sgn}(v)$ , där  $f$  är luftmotståndskoefficienten, funktionen  $\text{sgn}(v)$  är  $-1$  om  $v < 0$  och  $1$  annars. Funktionen `sgn` finns inte inbyggd i Python.

- B5F1 Det är rimligast att position och hastighet båda byter tecken när noll skärs. Om studsens sker mitt under tidssteget är en första approximation att bollen fortsätter uppåt i stället för nedåt, med samma fart. Att sätta positionen till noll eller bara byta tecken på ena eller andra ger klart sämre resultat.
- B5F2 Tidssteg bör ej hoppas över när noll är på väg att skäras. Positionen ska uppdateras med hastigheten för varje tidssteg.
- B5F3 Luftmotståndet ska vara motriktat nuvarande hastighet.
- B5F4 Koden ska aldrig riskera att skriva ut negativa positioner. Kontrollera om positionen är negativ efter positionsuppdateringen, inte i nästa anrop av `step`.
- B5F5 Metoden ska inte returnera något.
- B5F6 Om du använder funktionen `sgn` eller liknande måste den definieras. Det angavs uttryckligen i uppgiften att funktionen `sgn` inte finns inbyggd i Python i sig. Det finns en del andra funktioner som kan användas, men då måste du tydligt använda rätt namn och modul.
- B5F7 Luftmotståndet ska beräknas korrekt. Formeln är inte korrekt implementerad.
- B5F8 `step` ska inte avrunda. Simuleringen ska pågå så exakt det går, avrundning sker vid utskrift.

#### B6. Implementera `RollingBall.step`.

Bollen rullar med en initial hastighet åt något håll.

Den bromsas kontinuerligt av friktion.

Friktionsaccelerationen är proportionell mot normalaccelerationen och en friktionskoefficient. På ett plant underlag, som i vår uppgift, är normalaccelerationen likvärdig med tyngdaccelerationen, alltså 9.81m/s. Friktionsaccelerationen är givetvis motriktad den nuvarande hastigheten.

Se till att din `step`-metod får bollen att stanna till hastighet exakt 0 när friktionen har bromsat den tillräckligt – det är orimligt att friktionen får bollens rörelse att byta riktning. Motivera hur du gör detta.

Ledning: Accelerationen som verkar på bollen kan alltså beskrivas som  $a = -9.81f \operatorname{sgn}(v)$ , där  $f$  är friktionskoefficienten.

- B6F1 Inbromsning till 0 måste fungera oavsett föregående tecken på hastigheten. Det är fullt möjligt att skapa en instans av `RollingBall`.
- B6F2 Inbromsning bör utföras före uppdateringen av position, annars kommer bollen att backa något lite i sista tidssteget. Detta skulle undvikas.
- B6F3 Metoden ska inte returnera något.
- B6F4 Om du använder funktionen `sgn` eller liknande måste den definieras. Det angavs uttryckligen i uppgiften att funktionen `sgn` som i uppgiften inte finns inbyggd i Python i sig. Det finns en del andra funktioner som kan användas, men då måste du tydligt använda rätt namn och modul.
- B6F5 `step` ska inte avrunda. Simuleringen ska pågå så exakt det går, avrundning sker vid utskrift.
- B6F6 Friktionen ska beräknas korrekt. Friktionen är exempelvis inte proportionell mot hastighetens värde eller värdet i kvadrat. Den är konstant förutom att den inte kan leda till att hastigheten byter tecken, då den avstannar.