Danmarks Tekniske Universitet

Kursusnavn	Introduktion til programmering og databehandling
Kursusnummer	02631, 02632, 02633, 02634, 02692
Hjælpemidler	Alle hjælpemidler er tilladt
Varighed	2 timer
Vægtning	Opgaverne vægtes ens

Indhold

Aflevering

Du skal aflevere dine løsninger elektronisk:

1. Du kan uploade dine løsninger individuelt på CodeJudge

   https://dtu.codejudge.net/prog-jan20/assignments

   under *Exam*. Når du afleverer en løsning på CodeJudge bliver det test-eksempel som fremgår i opgavebeskrivelsen kørt på din løsning. Hvis din løsning består denne ene test, vil den fremgå som *Submitted*. Det betyder at din løsning består på dette ene test-eksempel. Du kan uploade til CodeJudge så mange gange som du ønsker under eksamen.

2. Du skal uploade alle dine løsninger på DTU's Onlineeksamen site. Hver assignment skal uploades som en separat .py fil med samme navn som funktionen i opgaven:

   (a) `riemann.py`
   (b) `packing.py`
   (c) `dice_sum_count.py`
   (d) `imputed_sum.py`
   (e) `tokenize.py`

   Filerne skal afleveres separat (*ikke* som en zip-fil) og skal have præcis disse filnavne.

Efter eksamen vil dine løsninger fra DTU Inside blive automatisk evalueret på CodeJudge på en række forskellige tests for at undersøge om de generelt fungerer korrekt. Bedømmelsen af din aflevering foretages udelukkende på baggrund af hvor mange af de automatiserede test der bestås.

- Du skal sikre dig at din kode følger specifikationerne nøje.
- Hver løsning må ikke indeholde nogen yderligere kode udover den specificerede funktion, dog kan `import`-sætninger inkluderes.
- Husk at du kan tjekke om dine løsninger følger specifikationerne ved at uploade dem til CodeJudge.
- Bemærk at alle vektorer og matricer anvendt som input eller output skal være numpy arrays.

Assignment A Riemann series

Betragt de konvergende rækker

$$s_a = \sum_{i=1}^{n} \frac{(-1)^{i+1}}{\lceil i/2 \rceil} = 1 - 1 + 1/2 - 1/2 + 1/3 - 1/3 + \ldots \quad (1)$$

$$s_b = \sum_{i=1}^{n} \frac{(-1)^{i+1}}{i} = 1 - 1/2 + 1/3 - 1/4 + 1/5 + \ldots \quad (2)$$

hvor $\lceil\ \rceil$ er ceil-funktionen. Vi vil gerne beregne summen efter n led fra en af rækkerne.

Problemdefinition

Skriv en funktion kaldet `riemann` som tager som input: n der at antallet af led og formen specificeret som enten "a" eller "b" svarende til henholdsvis ligning 1 eller ligning 2. Funktion skal returnere summen af rækken fra n antal led afhængig af formen.

Løsningsskabelon

```
def riemann(n, form):
  #insert your code
  return s
```

Input

`n`	Skalar med antallet af led (heltal, større end nul)
`form`	Streng der er enten "a" eller "b".

Output

`s`	Skalar med summen af rækken (flydende tal).

Eksempel

Betragt eksemplet hvor antallet af led er $n = 7$ og formen = "a"

$$s_a = \sum_{i=1}^{7} \frac{(-1)^{i+1}}{\lceil i/2 \rceil} = 1 - 1 + 1/2 - 1/2 + 1/3 - 1/3 + 1/4 \quad (3)$$

$$= 0 + 0 + 0 + 0.25 = 0.25. \quad (4)$$

Funktionen returnerer

$$\mathtt{s} = 0.25 \quad (5)$$

A

Assignment B Packing

Vi ønsker at teste om et to-dimensionelt firekantet objekt kan passe i en to-dimensionel kasse. Størrelsen af kassen og objektet er hver specificeret med en to-dimensionel vektor. Vi ønsker at vide størrelsen af manglende plads for hver dimension, der er differencen mellem objektstørrelsen og kassestørrelsen. Hvis der er nok plads i en dimension så skal funktionen returnere nul for den dimension. Objektet kan roteres og kun det tilfælde med bedst plads skal returneres.

Problemdefinition

Skriv en funktion kaldet `packing` der tager som input: En 2-dimensionel vektor med størrelse af kassen og en 2-dimensionel vektor med størrelsen af objektet. Funktionen skal returnere den manglende plads beregnet som differencen i hver dimension. Hvis der er nok plads i en dimension så skal den dimension sættes til nul. Objektet kan roteres og kun det tilfælde hvor der er den mindste sum skal returneres. Summen er beregnet som summen over elementer for den manglende plads. Hvis to summer er ens, så skal den ikke-roterede version foretrækkes.

Løsningsskabelon

```
def packing(box, object):
  # insert your code
  return d
```

Input

`box`	2-dimensionel vektor med størrelsesspecifikationen for kassen.
`object`	2-dimensionel vektor med størrelsesspecifikationen for objektet.

Output

`d`	2-dimensionel vektor med den manglende plads.

Eksempel

Betragt kassen, $\mathbf{b} = [2, 5]$, og objektet, $\mathbf{g} = [4, 2.1]$. Når den er roteret bliver størrelsen på objektet $\tilde{\mathbf{g}} = [2.1, 4]$. Den manglende plads er beregnet som $\mathbf{d}$ (for det ikke-roterede tilfælde) og $\tilde{\mathbf{d}}$ (for det roterede tilfælde):

$$\mathbf{d} = \mathbf{g} - \mathbf{b} = [4, 2.1] - [2, 5] = [2, -2.9] \tag{6}$$
$$\tilde{\mathbf{d}} = \tilde{\mathbf{g}} - \mathbf{b} = [2.1, 4] - [2, 5] = [0.1, -1] \tag{7}$$

Dimensioner hvor der er nok plads sættes til nul:

$$\mathbf{d}' = [2, 0] \tag{8}$$
$$\tilde{\mathbf{d}}' = [0.1, 0] \tag{9}$$

Summerne af elementerne af disse to vektorer er $2 + 0 = 2$ for det ikke-roterede tilfælde og $0.1 + 0 = 0.1$ for det roterede tilfælde, dvs. det roterede tilfælde er der hvor der er bedst plads, og den manglende plads der svarer til skal returneres som:

$$\mathtt{d} = [0.1, 0] \tag{10}$$

Assignment C Dice sum count

For et terningspil med én, to eller tre terninger vil vi tælle hvor mange gange summen af terningerne er lig med eller mindre end en given tærskelværdi når man tæller alle mulige kombination af terningerne.

Problemdefinition

Skriv en funktion kalde `dice_sum_count` der som input tager: Funktionen skal returnere antallet af terningekombinationer hvor summen er lig med eller mindre en given tærskelværdi.

Løsningsskabelon

```
def dice_sum_count(threshold, dice):
  # insert your code
  return count
```

Input	
`threshold`	Skalar med tærskelværdien (flydende tal)
`dice`	Skalar med antal terninger, enten 1, 2, eller 3 (heltal)

Output	
`count`	Skalar med antallet af summer lig med eller mindre end tærskelværdien (heltal)

Eksempel

Betragt tilfældet hvor tærskelværdien er 4 og hvor antallet af terninger er 2. De 36 forskellige kombinationer af terningekastet er:

$$[1,1],[1,2],[1,3],[1,4],[1,5],[1,6],[2,1],[2,2],[2,3],\ldots,[3,1],[3,2],\ldots,[6,5],[6,6] \tag{11}$$

Summen er hver af disse spil er

$$2,3,4,5,6,7,3,4,5,\ldots 4,5,\ldots 11,12 \tag{12}$$

Summerne der er mindre end eller lig med 4 er:

$$2,3,4,3,4,4, \tag{13}$$

Der er 6 summer mindre end eller lig med 4 og tallet 6 er returneret som

$$\texttt{count} = 6 \tag{14}$$

C

Assignment D Imputated sum

For en talrække vil vi beregne summen. Et problem med rækken er at nogle tal er "manglende" indikeret med tallet 999. De "manglende" tal skal beregnes som middelværdien af de umiddelbare nabotal.

Betragt en eksempel-talrække $\mathbf{x}$,

$$\mathbf{x} = [17, -4.1, 3, 999, 7, 18, 999] \tag{15}$$

Her er det 4. og det sidste element "manglende". Det 4. element beregnes som middelværdien af nabotallende 3 og 7, dvs. 5. og det sidste element sættes til naboværdien 18. Dette resulterer i en ny "imputeret" række:

$$\tilde{\mathbf{x}} = [17, -4.1, 3, 5, 7, 18, 18] \tag{16}$$

Summen af disse tal er 63.9.

Problemdefinition

Skriv en funktion `imputed_sum` som tager en vektor med tal og hvor manglende værdier er indikeret med værdien 999. De manglende værdier skal beregnes som middelværdien af de to umiddelbare nabotal. Hvis den manglende værdi er helt i starten af rækken eller helt i slutningen, så skal den manglende værdi sættes til den umiddelbare nabo. Rækken kan indeholde nul eller flere manglende værdier, men det kan antages at der ikke er to manglende værdier lige efter hinanden og at der altid er én ikke-manglende værdi.

Løsningsskabelon

```
def imputed_sum(x):
    #insert your code
    return s
```

Input

`x`	Vektor med talrække, hvor manglende værdier er indikeret med 999.

Output

`s`	Skalar med sum.

Eksempel

For eksemplet ovenover har vi

$$\mathbf{x} = [17, -4.1, 3, 999, 7, 18, 999] \tag{17}$$

Den resulterende sum skal returneres som

$$\mathbf{s} = 63.9 \tag{18}$$

D

Assignment E Tokenization

Givet en streng der repræsenterer en tekst fra et bretonsk-agtigt sprog vil vi finde antallet af individuelle ord. Vi ønsker at håndtere tilfældet med en apostrof (et enkelt-citationstegn) mellem "c" efterfulgt af "h", dvs. "c'h". Ved stedet med apostroffen mellem disse to tal skal ordet ikke splittes. For andre bogstavkombinationer med apostroffer skal strengen splittes ud i separate ord.

Problemdefinition

Skriv en funktion kaldet `tokenize` som tager en streng som input. Funktionens output skal returnere antallet af ord i strengen. Ord-adskillelsen er hvor der er mellemrum eller en apostrof, med undtagelse af når apostroffen er mellem "c" efterfulgt af "h". Det kan antages af alle tegn er enten bogstaverne fra a til z, mellemrumstegnet eller apostroffen. Det kan også antages at der aldrig er to mellemrum eller to apostroffer lige efter hinanden, at apostroffen altid er mellem bogstaver og at der altid er et eller flere ord.

Løsningsskabelon

```
def tokenize(text):
  # insert your code
  return words
```

Input	
`text`	Streng der repræsenterer en tekst.

Output	
`words`	Skalar med antal ord i teksten (heltal).

Eksempel

Betragt strengen

"`ma n'oc'h ket asur eus an lec'h`"

For dette eksempel splittes strengen ud til ordene

`"ma", "n", "oc'h", "ket", "asur", "eus", "an", "lec'h"`

Her er der 8 ord og det er returneret som

$$\texttt{words} = 8 \tag{19}$$

E