Danmarks Tekniske Universitet

Kursusnavn	Introduktion til programmering og databehandling Programmering og databehandling (andet programmeringssprog)
Kursusnummer	02632, 02633, 02634
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 (dtu.codejudge.net/prog-aug15/assignment) under *Afleveringer/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 CampusNet. Hver assignment skal uploades som en separat .py fil med samme navn som funktionen i opgaven:

   (a) `classifyBMI.py`
   (b) `cumulativeStock.py`
   (c) `RPNCalculator.py`
   (d) `starPoints.py`
   (e) `matrixCleanup.py`

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

Efter eksamen vil dine løsninger blive automatisk evalueret på CodeJudge på en række forskellige tests for at undersøge om de generelt fungerer korrekt. Bedømmelsen af din aflevering foretages udeklukkende på baggrund af hvor mange af de automatisered 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.
- 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 Body mass index

Body mass index (BMI) benyttes ofte til at klassificere undervægt, overvægt og fedme for voksne mennesker. Body mass index b er defineret som vægten w (i kilogram) divideret med kvadratet på højden h (i meter),

$$b = \frac{w}{h^2} \tag{1}$$

Den følgende tabel oplister navnene på forskellige BMI-grupper:

BMI group	BMI, b
`Severely underweight`	$b < 16$
`Underweight`	$16 \leq b < 18.5$
`Normal`	$18.5 \leq x < 25$
`Overweight`	$25 \leq x < 30$
`Obese`	$30 \leq x < 40$
`Severely obese`	$40 \leq x$

Problemdefinition

Skriv en funktion med navnet `classifyBMI` som tager højden og vægten som numeriske input og returnerer BMI-gruppen som en streng (skrevet præcis som i tabellen ovenfor.)

Løsningsskabelon

```
def classifyBMI(height, weight):
  #insert your code
  return BMIGroup
```

Input

`height`	Højde i meter (decimaltal).
`weight`	Vægt i kilogram (decimaltal).

Output

`BMIGroup`	BMI-gruppe (streng).

Eksempel

Hvis højden er $h = 1.85$ meter og vægten er $w = 88$ kilogram, kan BMI beregnes som $b = \frac{88}{1.85^2} = 25.71$ og strengen `Overweight` skal returneres.

A

Assignment B Stock management

Du arbejder med et lagerstyringssystem som holder styr på hvor mange enheder af en given vare en butik har på lager. Som input får du en liste af transaktioner, i form af en række heltal: Positive tal betyder at enheder tilføjes lageret og negative tal betyder at enheder fjernes fra lageret. Tallet nul betyder at lagerbeholdningen nulstilles. Før den første transaktion er antallet af enheder på lageret lig nul.

Problemdefinition

Skriv en funktion med navnet `cumulativeStock` som returnerer antallet af enheder på lager efter hver transaktion givet en vektor indeholdende listen af transaktioner.

Løsningsskabelon

```
def cumulativeStock(transactions):
  #insert your code
  return stock
```

Input	
`transactions`	Transaktioner (vektor af heltal).

Output	
`stock`	Antal enheder på lager efter hver transaktion (vektor af heltal).

Eksempel

Forestil dig følgende liste af transaktioner:

$$10, -4, -3, 10, -12, 0, 8$$

I begyndelsen er lagerbeholdningen 0, så efter første transaktion er der 10 enheder på lager. Ved anden transaktion fjernes 4 enheder, så lagerbeholdningen falder til 6 enheder. Derefter fjernes 3 enheder, så vi ender på 3 enheder på lageret. Herefter tilføjes 10 enheder, så vi kommer op på en lagerbeholdning på 13 enheder, hvorefter 12 enheder fjernes og vi er nede på 1 enhed på lageret. Ved den næste transaktion nulstilles lagerbeholdningen, og til slut tilføjes 8 enheder. Den samlede lagerbeholdning efter hver transaktion skal således være:

$$10, 6, 3, 13, 1, 0, 8$$

B

Assignment C Reverse Polish notation calculator

Reverse Polish (omvendt polsk) notation er en måde at skrive matematiske udtryk hvor de matematiske operatorer (såsom plus og minus osv.) står efter operanderne (de tal der regnes på). For eksempel kan det at lægge tallene 3 og 4 sammen udtrykkes som "`3 4 +`".

I denne opgave skal du lave en simpel omvendt polsk kalkulator som fungerer på følgende måde. Kalkulatoren har en liste af tal (kaldet en stack) som til at begynde med er tom. Hvis kalkulatoren modtager et tal, tilføjer den det for enden af stacken. Hvis kalkulatoren modtager en operation der skal udføres, udtrækker den de to sidste tal på stacken, udfører operationen på disse to tal, og tilføjer resultatet for enden af stacken. Kalkulatoren skal undstøtte følgende operationer:

	Operation
`+`	Addition
`-`	Subtraktion
`s`	Ombyt rækkefølge

Problemdefinition

Skriv en funktion med navnet `RPNCalculator` som tager som input en tekst-streng der definerer en række kommandoer i omvendt polsk notation. Kommandoerne er enten tal eller en af strengene `+`, `-`, `s`, og er adskilt af mellemrum. Funktionen skal returnere den endelige værdi af stacken (som en vektor) efter at have udført de givne operationer.

Løsningsskabelon

```
def RPNCalculator(commands):
  #insert your code
  return stack
```

Input	
`commands`	Sekvens af kommandoer (streng).

Output	
`stack`	Stackens værdi (vektor).

Eksempel

Forestil dig til eksempel følgende sekvens af kommandoer som input:

`2 10 17 - s`

Den første kommando tilføjer tallet 2 til stacken. Derefter tilføjes tallet 10 til stacken og efterfølgende tilføjes tallet 17. Stacken indeholder nu tallene 2, 10 og 17. Herefter arbejder operatoren `-` (subtraktion) på de to sidste tal (10 og 17), fjerner disse tal fra stacken, trækker 17 fra 10 hvilket giver -7, og lægger dette resultat på stacken, som dermed indeholder tallene 2 og -7. Endelig vil operatoren `s` (ombyt rækkefølge) få kalkulatoren til at ombytte de to sidste tal på stacken, som dermed indeholder tallene -7 og 2.

command:	2	10	17	-	s
stack:	2	2	2	2	-7
		10	10	-7	2
			17		

Det endelige resultat er dermed følgende vektor:

$$[-7,\ 2]$$

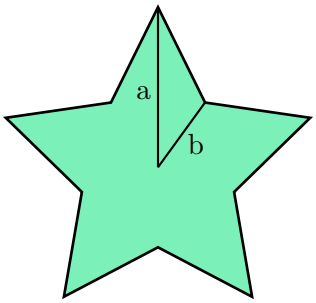
C

Assignment D Star area

Arealet af en n-takket stjerne er givet ved

$$A = n \cdot a \cdot b \cdot \sin\left(\frac{\pi}{n}\right) \tag{2}$$

hvor a og b er afstandene fra centrum til de ydre og indre hjørne-punkter på stjernen, og n er antallet af takker på stjernen. Her er en 5-takket stjerne illustreret:



Problemdefinition

Skriv en funktion med navnet `starPoints` som beregner det maksimale antal takker for en stjerne med de givne værdier af a og b, således at arealet af stjernen ikke er større end det angivne maksimale areal.

Løsningsskabelon

```
def starPoints(a, b, maxArea):
  #insert your code
  return points
```

Input	
`a`	Afstand fra centrum til ydre hjørne-punkter (positivt decimaltal)
`b`	Afstand fra centrum til indre hjørne-punkter (positivt decimaltal)
`maxArea`	Det maksimale areal af stjernen (positivt decimaltal)

Output	
`points`	Antal takker på stjernen (heltal)

Eksempel

Forestil dig at vi er givet $a = 2$, $b = 1$ og et maksimalt areal på 6.1. For forskellige værdier af n kan arealet beregnes som vist nedenfor:

n	3	4	5	6	7	8	9	10
A	5.196	5.657	5.878	6.000	6.074	6.123	6.156	6.180

Det største antal takker n for hvilket arealet ikke overstiger 6.1 er $n = 7$. Funktionen skal derfor returnere værdien 7.

D

Assignment E Removing rows and columns

Du arbejder med et datasæt i form af en rektangulær tabel af positive tal (en matrix M). Nogle af værdierne i tabellen er ukendte, og disse er markeret med værdien nul.

Problemdefinition

Skriv en funktion med navnet `matrixCleanup` som tager en matrix som input og returnerer en matrix hvor de rækker og kolonner som indeholder en eller flere nul-værdier er fjernet.

Løsningsskabelon

```
def matrixCleanup(M):
  #insert your code
  return MClean
```

Input

`M`	Input matrix (numpy array)

Output

`MClean`	Output matrix (numpy array)

Eksempel

Forestil dig følgende input matrix:

$$M = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 6 & 0 & 8 & 9 & 10 \\ 11 & 12 & 13 & 14 & 15 \\ 16 & 0 & 0 & 19 & 20 \end{bmatrix}$$

Da række 2 og 4 indeholder et eller flere nuller skal disse rækker fjernes, og da søjle 2 og 3 indeholder et eller flere nuller skal disse søjler også fjernes.

$$M = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ \cancel{6} & \cancel{0} & \cancel{8} & \cancel{9} & \cancel{10} \\ 11 & 12 & 13 & 14 & 15 \\ \cancel{16} & \cancel{0} & \cancel{0} & \cancel{19} & \cancel{20} \end{bmatrix}$$

Vi har da følgende matrix tilbage, som skal være output fra funktionen:

$$M_{\text{Clean}} = \begin{bmatrix} 1 & 4 & 5 \\ 11 & 14 & 15 \end{bmatrix}$$

E